

Lasso 7

Language Guide

blueworld

Trademarks

Lasso, Lasso Professional, Lasso Studio, LDML, Lasso Service, Lasso Connector, Lasso Web Data Engine, Blue World and Blue World Communications are trademarks of Blue World Communications, Inc. MySQL™ is a trademark of MySQL AB. All other products mentioned may be trademarks of their respective holders. See **Appendix C: Copyright Notices** in the Lasso Professional 7 Setup Guide for additional details.

Third Party Links

This guide may contain links to third-party Web sites that are not under the control of Blue World. Blue World is not responsible for the content of any linked site. If you access a third-party Web site mentioned in this guide, then you do so at your own risk. Blue World provides these links only as a convenience, and the inclusion of the links does not imply that Blue World endorses or accepts any responsibility for the content of those third-party sites.

Copyright

Copyright © 2004 Blue World Communications, Inc. This manual may not be copied, photocopied, reproduced, translated or converted to any electronic or machine-readable form in whole or in part without prior written approval of Blue World Communications, Inc.

Additional copies of this documentation may be purchased at the Blue World store at <http://store.blueworld.com/>.

Second Edition: January 14, 2004

Blue World Communications, Inc.
10900 NE 8th Street, Suite 900
Bellevue, Washington 98004 U.S.A.
Telephone: (425) 646-0288
Fax: (425) 454-4383
Email: blueworld@blueworld.com
Web Site: <http://www.blueworld.com>

Contents

Section I

Lasso Overview	19
----------------------	----

Chapter 1

Introduction	21
Lasso 7 Documentation	21
Lasso 7 Language Guide	22
Documentation Conventions	22

Chapter 2

Web Application Fundamentals	25
Web Browser Overview	25
Web Server Overview	31
HTML Forms and URL Parameters	32
Web Application Servers	34
Web Application Server Languages	35

Chapter 3

Format Files	37
Introduction	38
Storage Types	38
Naming Format Files	39

Character Encoding	40
Editing Format Files	40
Functional Types	41
Action Methods	42
<i>Table 1: Action Methods</i>	42
Securing Format Files	46
Output Formats	47
File Management	48
Specifying Paths	50
Format File Execution Time Limit	54

Chapter 4

LDML 7 Tag Language	55
Introduction	55
Syntax Types	56
<i>Table 1: LDML 7 Syntax Types</i>	56
Tag Types	60
<i>Table 2: LDML 7 Tag Types</i>	60
Tag Categories and Naming	67
<i>Table 3: LDML 7 Tag Categories</i>	67
<i>Table 4: LDML 7 Synonyms</i>	70
<i>Table 5: LDML 7 Abbreviations</i>	70
Parameter Types	71
<i>Table 6: Parameter Types</i>	71
Encoding	72
<i>Table 7: Encoding Keywords</i>	73
Data Types	74
<i>Table 8: Primary LDML 7 Data Types</i>	74
Expressions and Symbols	79
<i>Table 9: Types of LDML 7 Expressions</i>	79
<i>Table 10: Member Tag Symbol</i>	81
<i>Table 11: String Expression Symbols</i>	82
<i>Table 12: Math Expression Symbols</i>	83
<i>Table 13: Conditional Expression Symbols</i>	85
<i>Table 14: Logical Expression Symbols</i>	86
Delimiters	87
<i>Table 15: LDML 7 Delimiters</i>	87
<i>Table 16: HTML/HTTP Delimiters</i>	88
Illegal Characters	88
<i>Table 17: Illegal Characters</i>	89

Chapter 5

LDML 7 Reference	91
Overview	91
Figure 1: LDML 7 Reference	92
Search	93
Figure 2: Basic Search Page	93
Figure 3: Advanced Search Page	95
Figure 4: Comments Search Page	96
Figure 5: Examples Search Page	97
Browse	98
Figure 6: Category Tags Page	98
Figure 7: Legacy Tags Page	99
Detail	100
Figure 8: Tag Detail Page	100
Figure 9: Tag Comments Page	102
List	104
Figure 10: Preferred Tags Page	104
Figure 11: Legacy Tags Page	105

Section II

Database Interaction	107
----------------------------	-----

Chapter 6

Database Interaction Fundamentals ..	109
Inline Database Actions	109
Table 1: Inline Tag	110
Table 2: Inline Database Action Parameters	110
Table 3: Response Parameters	116
Action Parameters	120
Table 4: Action Parameter Tags	120
Table 5: [Action_Params] Array Schema	123
Results	124
Table 6: Results Tags	125
Showing Database Schema	125
Table 7: -Show Parameter	126
Table 8: -Show Action Requirements	126
Table 9: Schema Tags	127
Table 10: [Field_Name] Parameters	128
Table 11: [Required_Field] Parameters	129
SQL Statements	129
Table 12: SQL Inline Parameters	130

Table 13: -SQL Helper Tags	131
SQL Transactions	133

Chapter 7

Searching and Displaying Data135

Overview	135
Table 1: Command Tags	136
Table 2: Security Command Tags	139
Searching Records	139
Table 3: -Search Action Requirements	140
Table 4: Operator Command Tags	142
Table 5: Field Operators	143
Table 6: Results Command Tags	147
Finding All Records	150
Table 7: -FindAll Action Requirements	151
Finding Random Records	152
Table 8: -Random Action Requirements	152
Displaying Data	153
Table 9: Field Display Tags	154
Linking to Data	156
Table 10: Link Tags	157
Table 11: Link Tag Parameters	159
Table 12: Link URL Tags	160
Table 13: Link Container Tags	161
Table 14: Link Parameter Tags	162

Chapter 8

Adding and Updating Records171

Overview	171
Table 1: Command Tags	172
Table 2: Security Command Tags	174
Adding Records	174
Table 3: -Add Action Requirements	175
Updating Records	178
Table 4: -Update Action Requirements	178
Deleting Records	183
Table 5: -Delete Action Requirements	183
Duplicating Records	185
Table 6: -Duplicate Action Requirements	185

Chapter 9

MySQL Data Sources187

Overview	187
MySQL Tags	189
<i>Table 1: Enhanced MySQL Tags</i>	189
Searching Records	190
<i>Table 2: MySQL Search Field Operators</i>	190
<i>Table 3: MySQL Search Command Tags</i>	192
Adding and Updating Records	195
Value Lists	196
<i>Table 4: MySQL Value List Tags</i>	197
Creating Database Tables	202
<i>Table 5: Database Creation Tags</i>	203
<i>Table 6: [Database_CreateTable] Parameters:</i>	204
<i>Table 7: [Database_CreateField] and [Database_ChangeField]</i> <i>Parameters:</i>	206
<i>Table 8: MySQL Field Types</i>	207

Chapter 10

FileMaker Pro Data Sources211

Overview	212
Performance Tips	213
Compatibility Tips	214
FileMaker Pro Tags	215
<i>Table 1: FileMaker Pro Data Source Tag</i>	215
Primary Key Field and Record ID	216
Sorting Records	217
Displaying Data	218
<i>Table 2: FileMaker Pro Data Display Tags</i>	218
Value Lists	226
<i>Table 3: FileMaker Pro Value List Tags</i>	226
Images	231
<i>Table 4: Image Tags</i>	231
<i>Table 5: [IMG] Parameters</i>	233
FileMaker Pro Scripts	234
<i>Table 7: FileMaker Pro Scripts Tags</i>	234

Chapter 11**JDBC Data Sources237**

Overview 237

Using JDBC Data Sources 238

JDBC Schema Tags 239

*Table 1: JDBC Schema Tags240***Section III****Programming241****Chapter 12****Programming Fundamentals243**

Overview 244

Figure 1: Error Page244

Logic vs. Presentation 245

Data Output 247

Table 1: Output Tags247

Variables 249

*Table 2: Variable Tags250**Table 3: Variable Symbols250*

Data Types 253

Table 4: Data Type Tags253

Symbols 258

Member Tags 260

Forms and URLs 261

Chapter 13**Conditional Logic263**

If Else Conditionals 264

Table 1: If Else Tags265

Select Statements 267

Table 2: Select Tags267

Loops 268

*Table 3: [Loop] Tag Parameters269**Table 4: Loop Tags270*

Iterations 272

Table 5: Iteration Tags273

While Loops 273

Table 6: While Tags274

Abort Tag 274

Table 7: Abort Tag	274
Boolean Type	275
Table 8: Boolean Tag	275
Table 9: Boolean Symbols	275

Chapter 14

String Operations279

Overview	280
Table 1: String Tag	281
String Symbols	282
Table 2: String Symbols	282
String Manipulation Tags	285
Table 3: String Manipulation Member Tags	286
Table 4: String Manipulation Tags	287
String Conversion Tags	288
Table 5: String Conversion Member Tags	289
Table 6: String Conversion Tags	289
String Validation Tags	290
Table 7: String Validation Member Tags	290
To compare two strings:	291
Table 8: String Validation Tags	291
String Information Tags	291
Table 9: String Information Member Tags	292
Table 10: String Information Tags	293
Table 11: Character Information Member Tags	295
Table 12: Unicode Tags	297
String Casting Tags	297
Table 13: String Casting Member Tags	297
Regular Expressions	298
Table 14: Regular Expression Tags	298
Table 15: Regular Expression Matching Symbols	299
Table 16: Regular Expression Combination Symbols	300
Table 17: Regular Expression Replacement Symbols	300
Table 18: Regular Expression Advanced Symbols	301

Chapter 15

Math Operations305

Overview	305
Table 1: Integer Tag	306
Table 2: Decimal Tag	307
Mathematical Symbols	308
Table 3: Mathematical Symbols	308

Table 4: Mathematical Assignment Symbols	309
Table 5: Mathematical Comparison Symbols	310
Decimal Member Tags	311
Table 6: Decimal Member Tag	311
Table 7: [Decimal->SetFormat] Parameters	312
Integer Member Tags	313
Table 8: Integer Member Tags	313
Table 9: [Integer->SetFormat] Parameters	314
Math Tags	316
Table 10: Math Tags	316
Table 11: [Math_Random] Parameters	318
Table 12: Trigonometric and Advanced Math Tags	319
Locale Formatting	320
Table 13: Locale Formatting Tags	320

Chapter 16

Date and Time Operations321

Overview	321
Date Tags	322
Table 1: Date Substitution Tags	325
Table 2: Date Format Symbols	327
Table 3: Date Format Member Tags	329
Table 4: Date Accessor Tags	330
Duration Tags	331
Table 5: Duration Tags	332
Date and Duration Math	333
Table 6: Date Math Tags	334
Table 7: Date and Duration Math Tags	335
Table 8: Date Math Symbols	337

Chapter 17

Arrays and Maps339

Overview	339
Arrays	340
Table 1: Array Tag	340
Table 2: Array Member Tags	342
Table 3: [Array->Merge] Parameters	347
Maps	352
Table 4: Map Tag	352
Table 5: Map Member Tags	353
Pairs	356
Table 6: Pair Tag	357

Table 7: Pair Member Tags	357
Common Maps and Arrays	358
Table 8: Common Maps and Arrays	359

Chapter 18

Encoding	361
Overview	361
Encoding Keywords	365
Table 1: Encoding Keywords	365
Encoding Controls	366
Table 2: Encoding Controls	366
Encoding Tags	367
Table 3: Encoding Tags	367
Encryption Tags	368
Table 4: Encryption Tags	368
Compression Tags	371
Table 5: Compression Tags	371

Chapter 19

Sessions	373
Overview	373
Session Tags	375
Table 1: Session Tags	375
Table 2: [Session_Start] Parameters	376
Session Example	380

Chapter 20

Files and Logging	383
Includes	383
Table 1: Include Tags	386
Logging	388
Table 2: File Log Tags	389
Table 3: Lasso Error Log Tags	390
Table 4: Log Preference Tag	391
Table 5: Log Message Levels	391
Table 6: Log Destination Codes	392
File Tags	392
Table 7: File Tags	396
Table 8: Line Endings	401
File Uploads	402
Table 9: File Upload Tags	403

Table 10: [File_Uploads] Map Elements	403
File Streaming Tags	404
Table 11: [File] Tag	405
Table 12: File Open Modes	405
Table 13: File Read Modes	405
Table 14: File Streaming Tags	406

Chapter 21

Error Control	409
Overview	409
Error Messages	411
Figure 1: Built-In Error Message	411
Figure 2: Lasso Service Error Message	412
Figure 3: Authentication Dialog	412
Custom Error Page	412
Figure 4: Custom Error Page	413
Error Pages	414
Table 1: Error Response Tags	414
Error Tags	415
Table 2: Error Tags	415
Table 3: Error Type Tags	417
Error Handling	418
Table 4: Error Handling Tags	419

Chapter 22

Control Tags	425
Authentication Tags	425
Table 1: Authentication Tags	426
Administration Tags	428
Table 2: Administration Tags	428
Scheduling Events	431
Table 3: Scheduling Tag	432
Table 4: Scheduling Parameters	432
Process Tags	435
Table 5: Process Tags	435
Null Data Type	437
Table 6: Null Member Tags	437
Page Variables	439
Table 7: Page Variable Tags	439
Table 8: Page Variables	440
Configuration Tags	441
Table 9: Configuration Tags	441

Format File Execution Time Limit	442
<i>Table 10: Time Limit Tags</i>	442

Chapter 23

Miscellaneous Tags	443
Name Server Lookup	443
<i>Table 1: Name Server Lookup Tag</i>	443
Validation Tags	444
<i>Table 2: Valid Tags</i>	444
Unique ID Tags	445
<i>Table 3: Unique ID Tag</i>	445
Server Tags	445
<i>Table 4: Server Tags</i>	445

Chapter 24

LassoScript	447
LassoScript Overview	447
LassoScript Syntax	448
<i>Table 1: LassoScript Delimiters</i>	448

Section IV

Protocols and Media	453
---------------------------	-----

Chapter 25

Email	455
Sending Email	455
<i>Table 1: Email Tag</i>	456
<i>Table 2: [Email_Send] Parameters</i>	456

Chapter 26

Images and Multimedia	461
Overview	461
<i>Table 1: Tested and Certified Image Formats</i>	463
Casting Images as LDML Objects	464
<i>Table 2: [Image] Tag:</i>	464
<i>Table 3: [Image] Tag Parameters:</i>	464
Getting Image Information	465
<i>Table 4: Image Information Tags</i>	465
Converting and Saving Images	467
<i>Table 5: Image Conversion and File Tags</i>	467

Manipulating Images	468
Table 6: Image Size and Orientation Tags	469
Table 7: Image Effects Tags	470
Table 8: Annotate Image Tag	473
Table 9: Composite Image Tag	474
Table 10: Composite Image Tag Operators	474
Extended ImageMagick Commands	476
Table 11: ImageMagick Execute Tag	476
Serving Image and Multimedia Files	477
Table 12: Image Serving Tag	479

Chapter 27

HTTP/HTML Content and Controls ..483

Include URLs	484
Table 1: Include URL Tag	484
Table 2: [Include_URL] Parameters	485
Redirect URL	487
Table 3: Redirect URL Tag	487
HTTP Tags	488
Table 4: HTTP Tags	488
FTP Tags	489
Table 5: FTP Tags	489
Cookie Tags	491
Table 6: Cookie Tags	492
Table 7: [Cookie_Set] Parameters	493
Caching Tags	496
Table 8: [Cache] Tag	497
Table 9: [Cache] Tag Parameters	497
Table 10: LDML Object Cache Tags	500
Table 11: Cache Control Tags	501
Server Push	502
Table 12: Server Push Tag	503
Header Tags	503
Table 13: Header Tags	504
Request Tags	506
Table 14: Request Tags	507
Client Tags	508
Table 15: Client Tags	508
Server Tags	509
Table 16: Server Tags	509

Chapter 28

Wireless Devices511

Overview	511
Formatting WML	512
WAP Tags	515
<i>Table 1: WAP Tags</i>	515
WML Example	516

Chapter 29

XML519

Overview	520
XML Glossary	521
XML Data Type	522
<i>Table 1: XML Data Type Tag</i>	522
<i>Table 2: XML Member Tags</i>	523
XPath Extraction	525
<i>Table 3: [XML_Extract] Tag</i>	526
<i>Table 4: Simple XPath Expressions</i>	527
<i>Table 5: Conditional XPath Expressions</i>	529
XSLT Style Sheet Transforms	530
<i>Table 6: [XML_Transform] Tag</i>	531
XML Stream Data Type	532
<i>Table 7: XML Stream Data Type Tag</i>	532
<i>Table 8: XML Stream Node Types</i>	533
<i>Table 9: XML Stream Navigation Member Tags</i>	534
<i>Table 10: XML Stream Member Tags</i>	535
XML-RPC	537
<i>Table 11: [XML_RPCCall] Tag</i>	538
<i>Table 12: XML-RPC Built-In Methods</i>	539
<i>Table 13: XML-RPC and Built-In Data Types</i>	540
<i>Table 14: XML-RPC Data Type</i>	540
<i>Table 15: [XML_RPC] Call Tag</i>	540
<i>Table 16: [XML_RPC] Processing Tags</i>	543
SOAP	544
Serving XML	550
<i>Table 17: [XML_Serve] Serving Tags</i>	550
Formatting XML	550
XML Templates	552
<i>Table 18: FileMaker Pro XML Templates</i>	553
<i>Table 19: SQL Server XML Templates</i>	553

Chapter 30

Portable Document Format557

Overview	558
Creating PDF Documents	559
Table 1: [PDF_Doc] Tag and Parameters	559
Table 2: [PDF_Doc->Add] Tag and Parameters	562
Table 3: PDF Page Tags	563
Table 4: PDF Read Tags	564
Table 5: Page Insertion Tag and Parameters	565
Table 6: PDF Accessor Tags	566
Table 7: [PDF_Doc->Close] Tag	567
Creating Text Content	567
Table 8: PDF Font Tag and Parameters	568
Table 9: [PDF_Font] Member Tags	569
Table 10: [PDF_Text] Tag and Parameters	571
Table 11: [PDF_Doc->DrawText] Tag	573
Table 12: [PDF_List] Tags and Parameters	573
Table 13: Special Characters	575
Creating and Using Forms	575
Table 14: [PDF_Doc] Form Member Tags	576
Table 16: Form Placement Parameters	578
Creating Tables	582
Table 17: [PDF_Table] Tag and Parameters	582
Table 18: [PDF_Table] Member Tags	583
Table 19: Cell Content Tags	584
Creating Graphics	586
Table 20: [PDF_Image] Tag and Parameters	586
Table 21: [PDF_Doc] Drawing Member Tags	588
Creating Barcodes	590
Table 22: [PDF_Barcode] Tag and Parameters	590
Example PDF Files	592
Serving PDF Files	597
Table 23: PDF Serving Tags	598

Section V

Upgrading	601
-----------------	-----

Chapter 31

Upgrading Your Solutions	603
--------------------------------	-----

Introduction	604
Unicode Support	605
Bytes Type	607
<i>Table 1: Tags That Return the Bytes Type</i>	608
<i>Table 2: Byte and String Shared Member Tags</i>	609
<i>Table 3: Unsupported String Member Tags</i>	610
Syntax Changes (Lasso 6)	612
<i>Table 4: Syntax Changes</i>	612
Tag Name Changes (Lasso 5/6)	628
<i>Table 5: Unsupported Tags</i>	629
<i>Table 6: Tag Name Changes</i>	629
<i>Table 7: Deprecated Tags</i>	629
Syntax Changes (Lasso 5)	630
<i>Table 8: Syntax Changes</i>	630
Lasso MySQL (Lasso 5)	633
<i>Table 9: Lasso MySQL Syntax Changes</i>	634
Syntax Changes (Lasso WDE 3.x)	634
<i>Table 10: Syntax Changes</i>	634
<i>Table 11: Line Endings</i>	644
Tag Name Changes (Lasso WDE 3.x)	649
<i>Table 12: Command Tag Name Changes</i>	649
<i>Table 13: Substitution, Process, and Container Tag Name Changes</i>	650
Unsupported Tags (Lasso WDE 3.x)	651
<i>Table 14: Unsupported Tags</i>	651
FileMaker Pro (Lasso WDE 3.x)	652

Appendix A

LDML 7 Tag List	655
-----------------------	-----

LDML 7 Tag List	656
LDML 7 Legacy Tag List	676

Appendix B

Error Codes681

 Lasso Professional 7 Error Codes 681

Table 1: Lasso Professional 7 Error Codes682

 Lasso MySQL Error Codes 686

Table 2: Lasso MySQL Error Codes686

 FileMaker Pro Error Codes 690

Table 3: FileMaker Pro Error Codes690

 JDBC Error Codes 694

Table 4: JDBC Error Codes694

Appendix C

Index695

Section I

Lasso Overview

This section includes an introduction to the fundamental concepts and methodology for building and serving data-driven Web sites powered by Lasso 7. Every new user should read through all the chapters in this section.

- *Chapter 1: Introduction* includes information about the documentation available for Lasso 7 and about this book.
- *Chapter 2: Web Application Fundamentals* includes an introduction to essential concepts and industry terms related to serving data-driven Web sites.
- *Chapter 3: Format Files* discusses how to create and work with Lasso 7 format files.
- *Chapter 4: LDML 7 Tag Language* introduces the syntax of Lasso Dynamic Markup Language (LDML), the language of Lasso 7.
- *Chapter 5: LDML 7 Reference* introduces the reference database which contains complete details about the syntax of every tag in LDML 7.

After completing *Section 1: Lasso Overview* you can proceed to *Section II: Database Interaction* to learn how to store and retrieve information from a database and to *Section III: Programming* to learn how to program in LDML.

1

Chapter 1

Introduction

This chapter provides an overview of the Lasso 7 documentation, the section outline, and documentation conventions for this book.

- *Lasso 7 Documentation* describes the documentation included with Lasso 7 products.
- *Lasso 7 Language Guide* describes the sections in this book.
- *Documentation Conventions* includes information about typographic conventions used within the documentation.

Lasso 7 Documentation

The documentation for Lasso 7 products is divided into several different manuals and also includes several online resources. The following manuals and resources are available.

- **Lasso Professional 7 Setup Guide** is the main manual for Lasso Professional 7. It includes documentation of the architecture of Lasso Professional 7, installation instructions, the administration interface, and Lasso security. After the release notes, this is the first guide you should read.
- **Lasso 7 Language Guide** includes documentation of LDML (Lasso Dynamic Markup Language), the language used to access data sources, specify programming logic, and much more.
- **LDML 7 Reference** provides detailed documentation of each tag in LDML 7. This is the definitive reference to the language of Lasso 7. This reference is provided as a LassoApp and Lasso MySQL database within Lasso Professional 7 and also as an online resource from the Blue World Web site.

- **Extending Lasso Guide** is a collection of documentation and sample projects which provide instructions on how to extend Lasso.

Comments, suggestions, or corrections regarding the documentation may be sent to the following email address.

`documentation@blueworld.com`

Lasso 7 Language Guide

This is the guide you are reading now. This guide contains information about programming in LDML and is organized into the following sections.

- **Section I: Lasso Overview** contains important information about using and programming Lasso that all developers who create custom solutions powered by Lasso will need to know.
- **Section II: Database Interaction** contains important information about how to create format files that perform database actions. Actions can be performed in the internal Lasso MySQL database or in external MySQL, FileMaker Pro, or other databases.
- **Section III: Programming** describes how to program dynamic format files using LDML. This section covers topics ranging from simple data display through advanced error handling and alternate programming syntaxes.
- **Section IV: Protocols and Media** describes how to use Lasso to interoperate with other Internet technologies such as email servers and remote Web servers. It describes how to use Lasso to serve images and multimedia files. It also describes how to use Lasso to serve pages to various clients including Web browsers, WAP browsers and more.
- **Section V: Upgrading** contains information for users who are upgrading from or familiar with an earlier version of Lasso. This section details the differences between Lasso 7 and earlier versions of Lasso.
- **Appendices** contain listing of all LDML 7 tags and error codes.

Documentation Conventions

The documentation uses several conventions in order to make finding information easier.

Definitions are indicated using a bold, sans-serif type face for the defined word. This makes it easy to find defined terms within a page. Terms are defined the first time they are used.

Cross References are indicated by an italicized sans-serif typeface. For instance, the current section in this chapter is *Documentation Conventions*. When necessary, arrows are used to define a path into a chapter such as *Chapter 1: Introduction > Documentation Conventions*.

Code is formatted in a narrow, sans-serif font. Code includes HTML tags, LDML tags, and any text which should be typed into a format file. Code is represented within the body text (e.g., [Field] or <body>) or is specified in its own section of text as follows:

```
[Field: 'Company_Name']
```

Code Results represent the result after code is processed. They are indicated by a black arrow and will usually be the value that is sent to the client's Web browser. The following text could be the result of the code example above.

→ Blue World

Note: Notes are included to call attention to items that are of particular importance or to include comments that may be of interest to select readers. Notes may begin with **Warning**, **FileMaker Pro Note**, **IIS Note**, etc. to specify the importance and audience of the note.

To perform a specific task:

The documentation assumes a task-based approach. The contents following a task heading will provide step-by-step instructions for the specific task.

2

Chapter 2

Web Application Fundamentals

This chapter presents an overview of fundamental concepts that are essential to understand before you start creating data-driven Web sites powered by Lasso Professional 7.

- *Web Browser Overview* describes how HTML pages and images are fetched and rendered.
- *Web Server Overview* describes how HTTP requests and URLs are interpreted.
- *HTML Forms and URL Parameters* describes how GET and POST arguments are sent and interpreted.
- *Web Application Servers* describes how interactive content is created and served.
- *Web Application Server Languages* describes how commands can be embedded within a format file, processed, and served.

Web Browser Overview

The World Wide Web (WWW) is accessed by end-users through a Web browser application. Popular Web browsers include Microsoft Internet Explorer and Netscape Navigator. The Web browser is used to access pages served by one or more remote Web servers. Navigation is made possible via hyperlinks or HTML forms. The simple point-and-click operation of the Web browser masks a complex series of interactions between the Web browser and Web servers.

URLs

The location of a Web site and a particular page within a site are specified using a Universal Resource Locator (URL). All URLs follow the same basic format:

```
http://www.example.com:80/folder/file.html
```

The URL is comprised of the following components:

- 1 The **Protocol** is specified first, `http` in the example above and is followed by a colon. The World Wide Web has two protocols. HTTP (HyperText Transfer Protocol) which is for standard Web pages and is the default for most Web browsers and HTTPS (HyperText Transfer Protocol Secure) which is for pages served encrypted via the Secure Socket Layer (SSL).
- 2 The **Host Name** is specified next, `www.example.com` in the example above. The host name can be anything defined by a domain name registrar. It need not necessarily begin with `www`, the same server may be accessible using `example.com` or by an IP address such as `127.0.0.1`.
- 3 The **Port Number** follows the host name, `80` in the example above. The port number can usually be left off because a default is assumed based on the protocol. HTTP defaults to port `80` and HTTPS defaults to port `443`.
- 4 The **File Path** follows a forward slash, `/folder/file.html` in the example above. The Web server uses this path to locate the desired file relative to the root of the Web serving folder configured for the specified domain name. The root of the Web serving folder is typically `C:\inetpub\wwwroot\` for Windows 2000 servers and `/Library/WebServer/Documents` for Mac OS X servers.

HTTP Request

The URL is used by the Web browser to assemble an HTTP request which is actually sent to the Web server. The HTTP request resembles the header of an email file. It consists of several lines each of which has a label followed by a colon and a value.

Note: Most current Web browsers and Web servers support the HTTP/1.1 standard. Lasso Professional 7 also supports this standard. However, the examples in this book are written for the HTTP/1.0 standard in order to provide maximum compatibility with older Web browser clients.

The URL `http://www.example.com/folder/file.html` becomes the following HTTP request:

```
GET /folder/file.html HTTP/1.0
Accept: */*
Host: www.example.com
User-Agent: Web Browser/4.1
```

The HTTP request is comprised of the following components:

- 1 The first line defines the HTTP request. The action is **GET** and the path to what should be returned is specified `/folder/file.html`. The final piece of information is the protocol and version which should be used to return the data, `HTTP/1.0` in the example above.
- 2 The **Accept** line specifies the types of data that can be accepted as a return value. `*/*` means that any type of data will be accepted.
- 3 The **Host** line specifies the host which was requested in the URL.
- 4 The **User-Agent** line specifies what type of browser is requesting the information.

HTTP Response

Once an HTTP request has been submitted to a server, an HTTP response is returned. The response consists of two parts: a response header which has much the same structure as the HTTP request and the actual text or binary data of the page or image which was requested.

The URL `http://www.example.com/folder/file.html` might result in the following HTTP response header:

```
HTTP/1.0 200 OK
Server: Lasso Professional 7.0
MIME-Version: 1.0
Content-type: text/html; charset=iso-8859-1
Content-length: 7713
```

The HTTP response header is comprised of the following components:

- 1 The first line defines the type of response. The protocol and version are given followed by a response code, `200 OK` in the example above.
- 2 The **Server** line specifies the type of Web server that returned the data. `Lasso Professional 7` returns `Lasso Professional 7.0` in the example above.
- 3 The **MIME-Version** line specifies the version of the MIME standard used to define the remaining lines in the header.
- 4 The **Content-type** line defines the type of data returned. `text/html` means that ASCII text is being returned in HTML format. This line could also

read `text/xml` for XML data, `image/gif` for a GIF image or `image/jpeg` for a JPEG image.

The `charset=iso-8859-1` parameter specifies the character set of the page. Lasso returns pages in UTF-8 encoding by default or in the character set specified in the `[Content_Type]` tag.

5 Content-length specifies the length in bytes of the data which is returned along with this HTTP response header.

The header is followed by the text of the HTML page or binary data of the image which was requested.

Requesting a Web Page

The following are the series of steps which are performed each time a URL is requested from a Web server:

- 1 The Web browser determines the protocol for the URL. If the protocol is not HTTP then it might be passed off to another application. If the protocol is HTTPS then the Web browser will attempt a secure connection to the server.
- 2 The Web browser looks up the IP address of the server through a Domain Name Server (DNS).
- 3 The Web browser assembles an HTTP request including the path to the requested page.
- 4 The Web browser parses the HTML returned by the request and renders it for display to the visitor.
- 5 If the HTML contains any references to images or linked style sheets then additional HTTP requests with appropriate paths are generated and sent to the Web server.
- 6 The images and linked style sheets are used to modify the rendered HTML page.
- 7 Client-side scripting language such as JavaScript are interpreted and may further modify the rendered page.

The Web browser opens a new HTTP request for each HTML page, style sheet, or image file that is requested. All HTTP requests for a given HTML page can be sent to the same Web server or to different Web servers depending on how the HTML page is written. For example, many HTML pages reference advertisements served from a completely different Web server.

Character Sets

All Web pages must be transmitted from server to client using a character set that maps the actual bytes in the transmission to characters in the fonts used by the client's Web browser. The `Content-Type` header in the HTTP response specifies to the Web browser what character set the contents of the page has been encoded in.

Lasso processes all data internally using double-byte Unicode strings. Since two bytes are used to represent each character characters from single-byte ASCII are padded with an extra byte. Double-byte strings also allow for 4-byte or even larger characters using special internally encoded entities..

For transmission to the Web browser Lasso uses another Unicode standard UTF-8 which uses one byte to represent each character. UTF-8 corresponds roughly to traditional ASCII and the Latin-1 (ISO 8859-1) character set. Double-byte or 4-byte characters are represented by entities. For example, the entity `並` represents the double byte character `⌘`.

For older browsers or other Web clients it may be necessary to send data in a specific character set. Some clients may expect data to be transmitted in the pre-Unicode standard of Latin-1 (ISO 8859-1). Lasso will honor the `[Content_Type]` tag in order to decide what character set to use for transmission to the Web browser. Using the following tag will result in the Latin-1 (ISO 8859-1) character set being used.

```
[Content_Type: 'text/html; charset=iso-8859-1']
```

Note: UTF-8 is an abbreviation for the 8-bit (single-byte) UCS Transformation Format. UCS is in turn an abbreviation for Universal Character Set. Since 8-bit Universal Character Set Transformation Format is such a mouthful it helps to think of UTF-8 simply as the most common Unicode character encoding.

Cookies

Cookies allow small amounts of information to be stored in the Web browser by a Web server. Each time the Web browser makes a request to a specific Web server, it sends along any cookies which the Web server has asked to be saved. This allows for the Web server to save the state of a visitor's session within the Web browser and then to retrieve that state when the visitor next visits the Web site, even if it is days later.

Cookies are set in the HTTP header for a file that is sent from the Web server. A single HTML file can set many cookies and cookies can even be set in the headers of image files. Each cookie has a name, expiration date, value, and the IP address or host name of a Web server. The following line in an HTTP header would set a cookie named `session-id` that expired on

January 1, 2010. The cookie will be returned in the HTTP request for any domains that end in `example.com`.

```
Set-Cookie: session-id=102-2659358; path=/; domain=.example.com;
expires=Wednesday, 1-January-2010 08:00:00 GMT
```

Each time a request is made to a Web server, any cookies which are labeled with the IP address or host name of the Web server are sent along with all HTTP requests for HTML files or image files. The Web server is free to read these cookies or ignore them. The HTTP request for any file on `example.com` or `www.example.com` would include the following line.

```
Cookie: session-id=102-2659358
```

Cookies are useful because small items of information can be stored on the client machine. This allows a customer ID number, shopping cart ID number, or simple site preferences to be stored and retrieved the next time the user visits the site.

Cookies are dependent upon support from the Web browser. Most Web browsers allow for cookie support to be turned off or for cookies to be rejected on a case-by-case basis. The maximum size of cookies is Web browser dependent and may be limited to 32,000 characters or fewer for each cookie or for all cookies combined.

Cookies can be set to expire after a certain number of minutes or at the end of the current user's session (until they quit their Web browser). However, this expiration behavior should not be counted on. Some Web browsers do not expire any cookies until the Web browser quits. Others do not expire cookies until the machine hosting the Web browser restarts. Some Web browsers even allow visitors to alter the expiration dates of stored cookies.

Authentication

Web browsers support authentication of the visitor. A username and password can be sent along with each HTTP request to the server. This username and password can be read or ignored by the Web server. If the Web server is expecting a username and password and does not find any or does not find a valid username and password then the server can send back a challenge which forces the browser to display an authentication dialog box.

The following lines added to an HTTP response header will force most Web browsers to challenge the visitor for a username and password. The response code 401 Unauthorized informs the Web browser that the user is not authorized to view the requested file.

```
HTTP/1.0 401 Unauthorized
```

There are two ways for a visitor to authenticate themselves to a Web server. The first is through an authentication dialog box in response to a challenge by the Web server. The second is by specifying a username and password in a URL directly as follows:

```
http://username:password@www.example.com/folder/default.lasso
```

In either case, the username and password are transmitted to the Web server with each HTTP request in plain text unless a secure protocol such as HTTPS is used. The following line would be added to an HTTP request based on the URL above. The username and password are encoded, but are not encrypted.

```
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

The same username and password will continue to be transmitted to the Web server until the user re-authenticates or quits the Web browser application.

Note: See the section on *Authentication Tags* in *Chapter 22: Lasso Control Tags* for information about LDML tags that automatically prompt for authentication information.

Web Server Overview

The World Wide Web is served to end-users by Web server applications. Popular Web servers include Apache, WebSTAR, and Microsoft Internet Information Services (IIS). The Web server handles incoming HTTP requests for URLs from Web browsers. The interaction described in the previous section from the Web browser's point of view looks a little different from the Web server's point of view.

The following are the series of steps which are performed each time a URL is requested from a Web server:

- 1 The HTTP request is received on one of the ports which is being listened to by the Web server. Most Web servers listen on port 80 for HTTP requests and on port 443 for secure HTTPS requests.
- 2 The HTTP request is parsed and split into its components: protocol, host name, file path.
- 3 The host name is used to decide what virtual host to serve a Web page from. Most Web servers operate from a single IP address, but serve pages for several different domain names. These may be as simple as `www.example.com` and `example.com`.

- 4 The path to the page request is added to the server root for the specified virtual host. The virtual hosts may all start in a different folder on the hard drive.
- 5 The security settings of the server are checked to see if the user needs to be authenticated to receive the page they are requesting. If an appropriate username and password are not specified in the HTTP request then a challenge is sent in the HTTP response instead of the request page.
- 6 Server-side plug-ins or modules are called upon to process the request page. For example, requests for HTML pages that have a file name with the suffix `.lasso` will be sent to Lasso Service for processing. The processed page is returned to the Web server and may even be sent through multiple server-side plug-ins or modules before being served.
- 7 The requested HTML page or image is returned to the user with an appropriate HTTP response header.

HTML Forms and URL Parameters

HTML forms and URLs allow for significant amounts of data to be transmitted along with the simple HTTP requests defined in the previous sections. The data to be transmitted can either be included in the URL or passed in the HTTP request itself.

URL Parameters

A URL can include a series of name/value parameters following the file path. The name/value parameters are specified following a question mark `?`. The name and value are separated by an equal sign `=` and multiple name/value parameters are attached to a single URL with ampersands `&`. The following URL has two name/value parameters: `name1=value1` and `name2=value2`.

```
http://www.example.com/folder/file.lasso?name1=value1&name2=value2
```

The URL parameters are simply added to the file path which is specified in the HTTP request. The URL above might generate the following HTTP request. Since the parameters follow the word `GET` they are often referred to as `GET` parameters.

```
GET /folder/file.lasso?name1=value1&name2=value2 HTTP/1.0
Accept: */*
Host: www.example.com
User-Agent: Web Browser/4.1
```

Since the characters `:/?&=@#%` are used to define the structure of a URL, the file path and URL parameters cannot include these characters without modifying them so that the structure of the URL is not disturbed. The characters are modified by encoding them into `%nnn` entities where `nnn` is the hexadecimal ASCII code for the character being replaced. `/` is encoded as `%2f` for example.

HTML Forms

HTML forms provide user interface elements in the Web browser so that a visitor can customize the parameters which will be transmitted to the Web server along with an HTTP request. HTML forms can be used to modify the GET parameters of a URL or can be used to send POST parameters.

Note: A full discussion of the HTML tags possible within an HTML form is beyond the scope of this section. Please see an HTML reference for a full listing of HTML form elements.

Example of an HTML form with a GET method:

The following HTML form has an action which specifies the URL that will be returned when this form is submitted. In this case the URL is `http://www.example.com/folder/file.lasso`. The method of the form is defined to be GET. This ensures that the parameters specified by the HTML form inputs will be added to the URL as GET parameters.

```
<form action="http://www.example.com/folder/file.Lasso" method="GET">
  <input type="text" name="value1" value="value1">
  <input type="submit" name="value2" value="value2">
</form>
```

This form generates the following HTTP request. It is exactly the same as the HTTP request created by the URL `http://www.example.com/folder/file.lasso?name1=value1&name2=value2`.

```
GET /folder/file.lasso?name1=value1&name2=value2 HTTP/1.0
Accept: */*
Host: www.example.com
User-Agent: Web Browser/4.1
```

Example of an HTML form with a POST method:

The following HTML form has an action which specifies the URL that will be returned when this form is submitted. In this case the URL is `http://www.example.com/folder/file.lasso`. The method of the form is defined to be POST. This ensures that the parameters specified by the HTML form inputs will be added to the HTTP request as POST parameters and that the URL will be left unmodified.

```
<form action="http://www.example.com/folder/file.Lasso" method="POST">
  <input type="text" name="value1" value="value1">
  <input type="submit" name="value2" value="value2">
</form>
```

This form generates the following HTTP request. The request file is simply that which was specified in the action, but the method is now POST. The HTML form parameters are specified as the content of the HTTP request. They are still URL encoded, but now appear at the end of the HTTP request, rather than as part of the URL.

```
POST /folder/file.lasso HTTP/1.0
Accept: */*
Host: www.example.com
User-Agent: Web Browser/4.1
Content-type: application/x-www-form-urlencoded
Content-length: 27
value1=value1&name2=value2
```

HTML Forms and URL Responses

The GET and POST parameters passed in HTML forms or URLs are most often used by server-side plug-ins or modules to provide interactive or data-driven Web pages. The GET and POST parameters are how values are passed to Lasso in order to specify database actions, search parameters, or for any purpose a Lasso developer wants.

Web Application Servers

A **Web Application Server** is a program that works in conjunction with a Web server and provides programmatically generated HTML pages or images to Web visitors. Web application servers include programs that adhere to the Common Gateway Interface (CGI), programs which have built-in Web servers, plug-ins or modules for Web server applications, and services or daemons that communicate with Web server applications.

Lasso Professional 7 is a Web application server which runs as a background service and communicates with the Web server Apache via a module called Lasso Connector for Apache, the Web server WebSTAR V via a plug-in called Lasso Connector for WebSTAR, or IIS via an ISAPI filter called Lasso Connector for IIS.

Web application servers are triggered in different ways depending on the Web server being used. Many Web application servers are triggered based on file suffix. For example, all file names ending in .lasso could be processed by Lasso Service. Any file suffix can be configured to trigger

processing by Lasso Service including .html so all HTML pages will be processed before being served. Web application servers can usually also be set to process all pages that are served by a Web server.

Most Web application servers function by interpreting a programming or scripting language. Commands in the appropriate language are embedded in format files and then executed when an appropriate HTML form or URL is selected by a Web site visitor. The Web application server accepts the GET and POST parameters in the HTML form or URL, interprets the commands contained within the referenced format file, and returns a rendered HTML page to the Web site visitor.

Developers can choose to develop complete Web sites using the scripting language provided by a Web application server or they can purchase solutions which are written using the scripting language of a particular Web application server.

Lasso Professional 7 is a scriptable Web application server with a powerful tag-based language called Lasso Dynamic Markup Language (LDML). Custom solutions can be created by following the instructions contained in this Lasso 7 Language guide. Links to pre-packaged, third party solutions can be found on the Blue World Web site.

<http://www.blueworld.com/>

Web Application Server Languages

There are two main types of languages provided by Web application servers.

- **Scripting Languages** are used to specify programming logic and are generally close in function to traditional programming languages. Scripting languages can be used to assemble HTML pages and output them to the Web visitor. Server-Side JavaScript is an example of a scripting language.
- **Tag-Based Languages** are used to specify data formatting and programming logic within pre-formatted HTML or XML format files. The tags embedded in the format file are interpreted and the output is modified before the page is served to the Web visitor. Server Side Includes (SSI) is an example of a tag-based language.

Lasso Professional 7 provides one language, LDML, which functions as both a scripting language and a tag-based language. LDML tags can be used in LassoScripts as a scripting language to define programming logic. LassoScripts can be used to render individual HTML tags or to render

complete HTML documents programmatically. LDML tags can also be used as a tag-based language inside square brackets within HTML or XML code.

3

Chapter 3

Format Files

This chapter introduces the concept of format files that contain LDML tags. Understanding how to create and use format files is critical to understanding Lasso 7. All new users of Lasso 7 should read this chapter.

- *Introduction* includes basic information about how format files are created and used in Lasso 7.
- *Storage Types* introduces the different methods of storing and retrieving format files.
- *Naming Format Files* describes the rules for naming format files.
- *Character Encoding* describes how Lasso uses the Unicode byte order mark to determine whether to read a file using the UTF-8 or Latin-1 (also known as ISO 8859-1) character set.
- *Editing Format Files* explains the options which are available for editing format files.
- *Functional Types* describes the various ways in which format files are used and introduces functional names for different types of format files.
- *Action Methods* introduces the concept of actions and describes how format files and LDML interact to create an action.
- *Securing Format Files* explains the importance of maintaining security for your format files.
- *Output Formats* shows how to use a format file to create output of various types.
- *File Management* explains how the architecture of Lasso 7 influences where files are stored and how they can be manipulated.
- *Specifying Paths* shows how URLs, HTML forms, and paths can be used to refer to format files.
- *Format File Execution Time Limit* describes the built-in limit on the length of time that format files will be allowed to execute.

Introduction

Format files are text files that contain embedded Lasso 7 code. When a format file is processed by Lasso Service, the embedded LDML tags are interpreted, executed, and the results are substituted in place of the tags. The resulting document is then returned to the client. Web sites powered by Lasso 7 are programmed by creating format files which include user interface elements, database actions, and display logic.

This chapter describes the different methods of storing, naming, and editing format files. It also discusses how multiple format files and LDML work together to create actions. The chapter finishes with discussions of how to output different types of data with format files and how to reference format files from within LDML tags, URLs, and HTML forms.

Note: Many of the terms used in this chapter are defined in *Appendix A: Glossary* of the Lasso Professional 7 Setup Guide. Please consult this glossary if you are unsure how any words are being used in this language guide.

Storage Types

The term **Format File** is used to describe any text file that contains embedded Lasso 7 code. Format files are usually stored on the local disk of the machine which hosts a Lasso Web server connector, but can also be stored on a remote machine, the machine which hosts Lasso Service, or even in a database field.

Format files are always text-based, but the structure of the text is not important to Lasso. Lasso will find the embedded LDML 7 tags, process them, and replace them with the results. Lasso will not disturb the text that surrounds the LDML tags, but may modify text which is contained within LDML container tags. The most common types of format files are described below.

- **HTML Format Files** contain a mix of LDML tags and HTML tags. HTML format files can be edited in leading visual Web authoring programs with LDML tags represented as icons or displayed as plain text. The output is usually HTML suitable for viewing in a Web browser.
- **XML Format Files** contain a mix of LDML tags and XML tags. When a developer creates an XML format file it may not be strictly valid XML code. However, it is constructed in such a way that the output after being processed by Lasso is valid XML code. XML format files can be constructed so that their output conforms to any Document Type Definition (DTD) or XML Schema.

- **Text Format Files** contain a mix of LDML tags and ASCII text. Text format files can be used as the body of email messages or can be used to output data in any ASCII-compatible form.
- **LDML Format Files** contain only LDML tags. Pure LDML format files usually contain programming logic and include other content types as needed. A pure LDML format file could be a placeholder that returns the appropriate type of content to whatever client loads the page.

Lasso format files can be stored in a variety of locations depending on how they are going to be used. Four locations are listed below, along with brief descriptions of how format files stored within them are used.

- **Web Server** – Format files are typically stored as text files on the machine which hosts the Web serving software with a Lasso Web server connector. The format files are stored along with the HTML and image files that comprise the Web site. As the client browses the site, they may visit some pages which are processed by Lasso Service and others that are served without any processing.
- **Lasso Service** – Format files can be stored on the machine which hosts Lasso Service. Usually, these format files serve a special purpose such as library files in the LassoStartup folder that contain code which is executed when Lasso Service starts up.
- **Database Field** – Format files can be stored as text in a database field. When a database action is performed the contents of the field are returned to the client as if a disk-based text file had been processed and served. Permission must be granted in Lasso's administration interface in order to use a database field in this fashion. See *Chapter 7: Setting Up Data Sources* in the Lasso Professional 7 Setup Guide for more information.
- **Remote Server** – Lasso will not process LDML code which is stored on remote servers, but it can incorporate content from remote Web servers into the results served to the client or trigger CGI actions on remote servers using the [Include_URL] tag. See *Chapter 20: Files and Logging* for more information.

Naming Format Files

The Lasso Professional 7 Installer will automatically configure your Web server to pass files named with a .lasso suffix to Lasso Service for processing. Once it has finished processing a file, Lasso Service passes the resulting file back to the Web server, which in turn sends the file to the client's Web browser. Files with other extensions, such as .gif or .jpg image files or .html

files are served directly by the Web server without being processed by Lasso Service.

In addition, the Web server can be configured to send LDML format files with other extensions such as .xml or .wml to Lasso Service. It is even possible to configure the Web server to send all .html files to Lasso Service for processing. See *Chapter 6: Setting Global Preferences* and the configuration chapters in the Lasso Professional 7 Setup Guide for more information.

In order to promote the portability of your format files between Macintosh, Windows, and UNIX platforms, it is best to name them in a multi-platform friendly fashion. Never use reserved characters such as : ? & \ / # % ' " in file names. Avoid spaces, punctuation, stray periods, and extended ASCII characters. The safest file names contain only letters, numbers, and underscores. Some file systems are case-sensitive. Make sure that all references to a file are specified using the same case as the actual file name on disk. One option is to standardize on lowercase characters for all filenames.

Character Encoding

Lasso uses the standard Unicode byte order mark to determine if a format file is encoded in UTF-8. If no byte order mark is present then the format file will be assumed to be encoded using the Macintosh (or Mac-Roman) character set on Mac OS X or the Latin-1 (or ISO 8859-1) character set on Windows or Linux. Lasso does not support UTF-16 or UTF-32 format files.

Standard text editors such as Bare Bones BBEdit can save files using UTF-8 encoding with the byte order mark included. Consult the manual for the text editor to see how to change the encoding of format files and how to include the proper byte order mark to specify the encoding.

Note: It is recommended to use the Macintosh or Latin-1 character set only for format files that do not contain extended, accented, or foreign characters.

Editing Format Files

Lasso format files can be edited in any text editor. If a format file contains markup from a specific language such as HTML, WML, or XML then it can be edited using an application which is specific to creating that type of file.

In order to make creating and editing Lasso format files which contain HTML easier, Blue World supplies a product called Lasso Studio. Lasso Studio provides tag-specific inspectors, wizards, and builders which allow a developer to quickly build Lasso format files within either Macromedia

Dreamweaver, or Adobe GoLive. More information about Lasso Studio is available at the following URL:

<http://www.LassoStudio.com/>

To ease editing of Lasso format files within leading text editors such as Bare Bones BBEEdit or Macromedia Home Site consult the Lasso Solutions page at the following URL for links to various third-party solutions:

<http://www.blueworld.com/blueworld/products/lassosolutions.html>

Functional Types

Format files can be classified based on the types of LDML tags they contain or based on the commands they will perform within a Web site. The following list contains terms commonly used to refer to different types of format files. A format file can be classified as being one or more of these types.

- **Pre-Lasso** is used to refer to a format file that contains only command tags within HTML form inputs and URLs. Since Lasso does not perform any substitutions on command tags, these format files do not require any processing by Lasso before they are served to a client. Pre-Lasso format files can be named with a .html file name extension and can even be served from a Web server that does not have a Lasso Web server connector installed.
- **Post-Lasso** format files are the most common type of format files. Post-Lasso format files can contain any combination of tags in square brackets, command tags in HTML form inputs and URLs, and LassoScripts. Post-Lasso format files need to be processed by Lasso Service before they are served to the client. They are usually named with a .lasso file name extension.
- **Library** format files are used to modify Lasso's programming environment by defining new tags and data types, setting up global constants, and performing initialization code. Libraries are included in other format files to modify the environment in which a single format file is processed or loaded at startup to modify the global environment in which all format files are processed.
- **Add Page, Search Page, Update Page, Listing Page, Detail Page** and others are format file names based upon the action which the client will perform when they load the page in their Web browser. For example, a format file might implement the search page of a site. An update page would allow a user to edit a record from a database. A listing page is

usually the result of a search and contains links to a detail page which presents more information about each of the records listed.

- **Add Response, Search Response, Delete Response** and others are format files named based on the action which results in the format file being served to the client. These are typically called response pages. For example, a delete response is served in response to the client opting to delete a record from the database.
- **Error Page, Add Error, Search Error** and others are format files that provide an error message to the client based on the current action.

Action Methods

Web servers and Lasso Service are passive by nature. The software waits until an action is initiated by a client before any processing occurs. Every page load which is processed by Lasso can be thought of as an action with two components: a source and a response. A visitor selects a URL or submits an HTML form within the source format file and receives the response format file. The different types of Lasso actions are summarized in the table below and then described in more detail in the sections that follow.

Table 1: Action Methods

Action Method	Example
URL Action	<code>http://www.example.com/default.lasso</code>
HTML Form Action	<code><form action="Action.Lasso" method="post"> ... </form></code>
Inline Action	<code>[Inline: -Database='Contacts', ..., -Search] ... [/Inline]</code>
Scheduled Action	<code>[Event_Schedule: -URL='default.lasso', -Delay='10']</code>
Startup Action	<code>/LassoStartup/startup.lasso</code>

URL Action

A URL action is initiated or called when a client selects a URL in a source file. The source file could be an HTML file from the same Web site, an HTML file from another Web site, the “favorites” of a Web browser, or could be a URL typed directly in a Web browser. The selected URL triggers a designated response file that is processed and returned to the client.

The characteristics of the URL determine the nature of the action which is performed.

- **HTML** – If the URL references a file with a .html file name extension then no processing by Lasso will occur (unless the Web server has been configured to send .html files to Lasso Service.). The referenced HTML file will be returned to the client unchanged from how it is stored on disk.
`http://www.example.com/default.html`
- **Lasso** – If the URL references a file with a .lasso file name extension then Lasso Service will be called upon to process the file. The referenced format file will be returned to the client after Lasso Service has evaluated all the LDML tags contained within.
`http://www.example.com/default.lasso`
- **Action.Lasso** – If the URL references Action.Lasso then any command tags contained in the URL will be evaluated and an appropriate response will be returned to the user. The response to an Action.Lasso URL will always be processed by Lasso Service whether it is a .html file, a .lasso file, or a database field.
`http://www.example.com/Action.Lasso?-Response=default.html`

Note: Lasso will only process files with extensions that have been registered within Lasso Administration. See *Chapter 6: Setting Global Preferences* of the Lasso Professional 7 Setup Guide for more information.

HTML Form Action

An HTML form action is initiated or called when a client submits an HTML form in a source file. The source file could be an HTML file from the same Web site or an HTML file from another Web site. The form action and inputs of the form are evaluated and trigger a designated response file that is processed and returned to the client.

The characteristics of the form action determine the nature of the action which is performed.

- **Lasso** – If the HTML form references a file with a .lasso file name extension then Lasso Service will be called upon to process the file. The referenced format file will be returned to the client after Lasso Service has evaluated all the LDML tags contained within the inputs of the form.
`<form action="default.lasso" method="post">`
`...`
`</form>`
- **Action.Lasso** – If the HTML form references Action.Lasso then any command tags contained in the inputs in the form will be evaluated and

an appropriate response will be returned to the user. The response to an HTML form with an Action.Lasso form action will always be processed by Lasso Service whether it is a .html file, a .lasso file, or a database field.

```
<form action="Action.Lasso" method="post">
  <input type="hidden" name="-Response" value="default.lasso"
  ...
</form>
```

Note: Lasso will only process files with extensions that have been registered within Lasso Administration. See *Chapter 6: Setting Global Preferences* of the Lasso Professional 7 Setup Guide for more information.

Inline Action

Inline actions are initiated when the format file in which they are contained is processed by Lasso Service. The result of an inline action is the portion of the format file contained within the [Inline] ... [/Inline] tags that describe the action. As with all Lasso format files, inline actions are processed as the result of a URL being visited or an HTML form being submitted. However, inline actions are not reliant on command tags specified in the URL or HTML form.

- **Inline Tag** – The [Inline] ... [/Inline] container tags can be used to implement an inline action within a format file. The action described in the opening [Inline] tag is performed and the contents of the [Inline] ... [/Inline] tags is processed as a sub-format file specific to that action.

```
[Inline: ... Action Description ...]
... Response ...
[/Inline]
```

- **Multiple Inlines** – A single format file can contain many [Inline] ... [/Inline] container tags. Each set of tags is implemented in turn. A single format file can be used to perform many different database actions in different databases as the result of a single URL action or HTML form action.

```
[Inline: ... Action One Description ...]
... Response One ...
[/Inline]
```

```
[Inline: ... Action Two Description ...]
... Response Two ...
[/Inline]
```

- **Nested Inlines** – Inlines can be nested so that the results of one inline action are used to influence the processing of subsequent inline actions. Nested inline actions allow for complex processing to be performed such

as copying records from one database to another or summarizing data in a database.

```
[Inline: ... Action One Description ...]
[Inline: ... Action Two Description ...]
... Combined Response ...
[/Inline]
[/Inline]
```

- **Named Inlines** – Inlines can be processed at the top of a format file and their results can be used later in the format file. This allows the logical processing of an action to be separated from the data formatting. The results of the inline action are retrieved by specifying the inline's name in the [Records] ... [/Records] container tag.

```
[Inline: -InlineName='Action', ... Action Description ...]
... Empty ...
[/Inline]
...
[Records: -InlineName='Action']
... Response ...
[/Records]
```

Scheduled Action

Scheduled actions are initiated when they are queued using the [Event_Schedule] tag in a source file. The source file could be a format file which is loaded as the result of an action by a client or could be loaded as a startup action. The response to the scheduled action is not processed until the designated date and time for the action is reached.

Any type of format file can be called as a scheduled action, but the results will not be stored. Scheduled format files can effectively be thought of as pure LDML format files. Scheduled format files can use logging or email messages to notify a client that the action has occurred. See *Chapter 22: Control Tags* for more information.

- **Lasso** – The URL referenced when the action is scheduled will usually contain a .lasso file name extension. The referenced format file will be processed when the designated date and time is reached, but the results will not be returned to any client. For example, the following [Event_Schedule] tag schedules a call to a page that will send an email report to the administrator of the site every 24 hours (1440 minutes), even after server restarts:

```
[Event_Schedule: -URL='http://www.example.com/admin/emailreport.lasso',
-Delay='1440', -Repeat=True, -Restart=True]
```

Startup Action

Startup actions are initiated when Lasso Service is launched by placing format files in the `LassoStartup` folder. Format files which are processed at startup are library files which are used to set up the global environment in which all other pages will be processed. For example, they can add tags and custom data types to the global environment, set up global constants, or queue scheduled actions.

- **Lasso** – Format files with `.lasso` file name extensions are used at startup to queue scheduled actions or perform routine tasks on the databases or files managed by Lasso Service. Any format files in the `LassoStartup` folder will be processed every time Lasso Service is launched.
- **Library** – Libraries of LDML tags and custom data types can be processed at startup in order to extend the global environment in which all other pages are processed. All LDML tags and data types in a library processed at startup will be available to all other format files processed by Lasso Service. See *Chapter 20: Files and Logging* for more information about libraries.

Securing Format Files

The information being collected or served in a Web site is often of a sensitive nature. Credit card numbers and visitor's personal information must be kept secure. Proper format file security is the first step toward creating a Web site which only provides the information you want it to publish.

The LDML code contained in a format file should be secured so visitors cannot examine it. Format files contain information about how to access your databases. They may contain passwords, table and field names, or custom calculations.

LDML code in a format file is implicitly secured if it is stored in a format file with a `.lasso` file extension. The code in the file will always be processed by Lasso before it is served to visitors. Visitors can access the HTML source of the file they receive, but cannot access the LDML source of the original format file.

It is important to ensure that your format files cannot be accessed unsecurely through other Internet technologies such as FTP, Telnet, or file sharing. Make sure that the files in your Web serving folder can only be accessed by trusted developers and administrators. See *Chapter 8: Setting Up Security* in the Lasso Professional 7 Setup Guide for more information.

Output Formats

Although Lasso format files are always text files, they can be used to output a wide variety of different data formats. The most basic format files match the output format. For example, HTML format files are used to return HTML output to Web browsers. But, pure LDML format files can be used to return data in almost any format through the use of the [include] tag and data from database fields.

This section describes how to output the most common data formats from Lasso format files.

Text Formats

Lasso can be used to output any text-based data format. Format files are usually based on a file of the desired type. The following are common output formats:

- **HTML** is the most common output format. Usually, HTML output is generated from HTML format files. The embedded LDML tags are processed, altering and adding to the content of the file, but the essential characteristics of the file remain unchanged.
- **XML** is rapidly becoming a standard for data exchange on the Internet. XML output is usually generated through Lasso by processing XML format files. The embedded LDML tags are processed, altering and adding content to the XML data in the file. The resulting XML data can be made to conform to any Document Type Definition (DTD) or XML Schema desired.
- **WML** is the language used to communicate with WAP-enabled wireless devices. WML is a language which is based on XML. It is an example of a DTD or XML Schema to which output data must conform. Lasso usually generates WML output by processing WML format files. Developers can create WML format files by using a WML authoring tool and then embedding LDML tags within.
- **PDF** or Portable Document Format is Adobe's machine-independent format for distribution of electronic documents. Lasso can be used in concert with PDFs in several ways. Lasso can be used to process forms embedded within PDF files and to return results to a client. Lasso can be used to generate ASCII PDFs through custom programming. Finally, Lasso can be used to provide access control to PDFs so only authorized users are able to download certain PDFs.

Binary Formats

Lasso can be used to output a variety of binary data formats. Generally, Lasso is not used to perform any processing on the binary data being served, but is just a conduit through which pre-existing binary data is routed. See *Chapter 26: Images and Multimedia* for more information about each of these methods. The following list describes common methods of outputting binary data.

- **URLs** can be created and manipulated using LDML. For example, a database could contain a file name in a field. LDML can be used to convert that file name into a valid URL which will then be served as part of an HTML page. The binary data will be fetched from the client directly without any further action by Lasso.
- **Database Fields** can be used to store binary data such as image files in a container or binary format. If a Lasso data source connector for the appropriate database supports fetching binary data, then Lasso can serve the binary data or image files directly from the database field using the [Field], [Image_URL] or -Image tags.
- **Binary Files** can be served through Lasso using a combination of the [Include_Raw] tag to output the binary data and the [Content_Type] tag to report to the client what type of data is being served.

File Management

Lasso 7 introduces a new distributed architecture. Lasso Service can be installed on one machine and a Lasso Web server connector can be installed into a Web server on a different machine. It is important to realize where format files are stored so they can be located on the appropriate machine.

Note: In most Lasso 7 installations Lasso Service and a Lasso Web server connector will be installed on the same machine. The discussion below still applies since the various components of Lasso 7 will operate out of different folders. An administrator can set up a machine so the same files are shared by all components of Lasso.

Lasso Web Server Connector

Most format files for a Web site will be stored on the same machine as a Lasso Web server connector in the Web serving folder which contains the HTML and image files for the Web site.

- **Client Format Files** are stored alongside the HTML and image files which comprise a Web site. To the client, these format files appear no different from plain HTML files except that they contain dynamic data.
- **Included Files** are stored in the Web serving folder. These are files which are incorporated into format files using the `[Include]` and `[Include_Raw]` tags. Included files could be other format files, plain HTML files, images files, PDF files, etc.
- **Library Files** can be stored in the Web serving folder. These files contain definitions for LDML tags and data types. Library files are referenced much like included files. The custom tags and data types defined in the library file are available only in the pages which load the library file.
- **Administrative Files** are stored in the Web serving folder in a folder named Lasso. These files comprise the Web-based administration interface for Lasso Service.

Lasso Service

Format files which are stored on the same machine as Lasso Service are used primarily when Lasso Service starts up to set up the global environment. However, other files which are manipulated by Lasso's logging and file tags are also stored on the Lasso Service machine.

- **Startup Format Files** are stored in the LassoStartup folder with Lasso Service. These files are processed when Lasso Service is launched and can perform routine tasks or modify the global environment in which all other Lasso format files will be processed. Any LDML tags, data types, or global constants defined in these libraries will be available to all pages which are processed by Lasso Service.
- **Startup LassoApps** are stored in the LassoStartup folder with Lasso Service. The default page of each LassoApp is processed at startup and the LassoApp is pre-loaded into memory for fast serving.
- **Log Files** are created using the `[Log]` tag. These files can be used to store information about the format files which have been processed by Lasso Service. Log files are created on the same machine as Lasso Service.
- **Uploaded Files** are stored in a temporary location in a folder with Lasso Service. Files can be uploaded by a client using a standard HTML file input. Uploaded files must be moved from their temporary location to a permanent folder before the page on which they were uploaded finishes processing.
- **File Tags** operate on files in folders on the same machine as Lasso Service. The file tags can be used to manipulate log files or uploaded files. The file tags are also used to manipulate HTML and other format

files in the Web serving folder if Lasso Service is installed on the same machine as a Lasso Web server connector or if file sharing between the two machines facilitates accessing the files as a remote volume. See *Chapter 20: Files and Logging* for more information.

Note: A user can only access files to which the group they belong has been granted access. See *Chapter 8: Setting Up Security* in the Lasso Professional 7 Setup Guide for more information.

Database

Format files can be stored in any database which is available to Lasso Service. They can be stored in the local Lasso MySQL database or in a remote database hosted on another machine.

- **Format Files** stored in database fields can be included in a page using the [Process] tag. In the following example, the field LDML_Template is processed using the [Process] tag:

```
[Process: (Field: 'LDML_Template')]
```

Database fields can also be referenced through appropriate URL or HTML form parameters. See *Chapter 7: Setting Up Data Sources* in the Lasso Professional 7 Setup Guide for more information about granting permission to use a field as a format file. In the following example, the field LDML_Template is used to format the response to the URL:

```
http://www.example.com/Action.Lasso?-Response=Field:LDML_Template
```

Specifying Paths

Format files can be referenced in many different ways depending on how they are being used. They can be referenced in any of the following ways:

- A URL can be used to reference a format file with a .lasso file extension directly:

```
http://www.example.com/default.lasso
```

- A URL can be used to reference format files with any file extensions by calling Action.Lasso and then specifying the format file in a -Response command tag:

```
http://www.example.com/Action.Lasso?-Response=default.html
```

- An HTML form can be used to reference a format file with a .lasso file extension directly in the form action:

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

- An HTML form can be used to reference format files with any file extensions by calling Action.Lasso as the form action and then specifying the format file in a -Response hidden input:

```
<form action="Action.Lasso" method="post">
  <input type="hidden" name="-Response" value="default.html">
  ...
</form>
```

- A format file can be referenced from within certain LDML tags. For instance, the [Include] tag takes a single format file name as a parameter:


```
[Include: 'default.lasso']
```

Paths are specified for format files differently depending on what type of format file contains the path designation and to which type of format file is being referred.

Note: Lasso cannot be used to reference files outside of the Web server root unless specific permission has been granted within Lasso Administration. See *Chapter 8: Setting Up Security* in the Lasso Professional 7 Setup Guide for more information.

Relative and Absolute Paths

Most paths in Lasso format files follow the same rules as the paths between HTML files served by the Web server. Relative and absolute paths are interpreted either by the client's Web browser or by Lasso Service. These paths are all defined within the context of the Web serving folder established by the Web server which is hosting a Lasso Web server connector. If a single Web server is used to host multiple sites, the Web serving folder could be different for each virtual host.

- **Relative Paths** between files can be specified using all the rules and features of URL file paths. For example, the following anchor tag designates a response in the same folder as the current page:

```
<a href="response.lasso">Response</a>
```

- Paths can use ../ to specify a higher level folder. The following anchor tag designates a response in the folder one level higher than that which contains the current page:

```
<a href="../response.lasso">Response</a>
```

- Relative paths designated within LDML tags follow the same basic rules except that ../ cannot be used to access the parent folder for a format

file. For example, the following [Include] tag includes a file from the same folder as the current page.

```
[Include: 'include.lasso']
```

- **Absolute Paths** are referenced from the root of the Web serving folder as designated by the Web serving software. The Web server root is specified using the / character. The following anchor tag designates a response file contained at the root level of the current Web site:

```
<a href="/response.lasso">Response</a>
```

- Absolute paths designated within LDML tags work the same as absolute paths in URLs. The following [Include] tag includes a file contained at the root level of the current Web site.

```
[Include: '/include.lasso']
```

For more information about specifying relative and absolute paths, consult your favorite HTML reference or the documentation for your Web serving application.

Action.Lasso Paths

If a format file has been called using Action.Lasso in either a URL or in an HTML form action then all paths within the format file will be evaluated relative to the stated location of Action.Lasso.

- Action.Lasso could be specified as Action.Lasso so it appears to be located in the same folder as the calling format file. All paths must then be specified as if the referenced format file was located in the same folder as the calling format files. Paths relative to the referenced format file will fail, but paths relative to the calling format file will succeed.

```
<a href="Action.Lasso?-Database=Contacts& ... ">Response</a>
```

- Action.Lasso could be specified as /Action.Lasso so it appears to be located at the root of the Web serving folder. All paths must then be specified as if the referenced format file was located at the root of the Web serving folder. Paths relative to the referenced format file will fail.

```
<a href="/Action.Lasso?-Database=Contacts& ... ">Response</a>
```

- Action.Lasso can also be specified using an arbitrary path such as /Folder/Action.Lasso. In this case all paths will be relative to the specified location of Action.Lasso.

```
<a href="/Folder/Action.Lasso?-Database=Contacts& ... ">Response</a>
```


Database Field Paths

The path to database fields which are going to be used as format files is a special case since these files are not contained on the local disk of the Web serving machine.

- In a URL, a field named `Example_Template` can be referenced as follows. Usually, `Action.Lasso` is used as the target of a URL and the field is specified in a `-Response` command tag. The URL must contain a valid database action that returns a record from which the field will be used. The following example searches for a person from the `Contacts` database whose ID is 1. The value of `Example_Template` for that person is used as the response format file.

```
http://www.example.com/Action.Lasso?-Database=Contacts&-Table=People&-KeyField=ID&-KeyValue=1&-Search&-Response=Field:Example_Template
```

- In an HTML form, a field named `ExampleTemplate` can be referenced as follows. Usually, `Action.Lasso` is used as the form action and the field is specified in a hidden input using a `-Response` command tag. This form uses the same database action defined in the URL above.

```
<form action="Action.Lasso" method="post">
  <input type="hidden" name="-Search" value="">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-KeyValue" value="1">
  <input type="hidden" name="-Response" value="Field:Example_Template ">
</form>
```

Note: Permission must be granted in Lasso Administration for a field to be used as a response field. See *Chapter 7: Setting Up Data Sources* in the Lasso Professional 7 Setup Guide for more information.

Lasso Service Paths

Paths to format files on the machine hosting Lasso Service are specified differently than those which are used in format files on the machine hosting a Lasso Web server connector. Format files on the machine hosting Lasso Service are usually only referenced by the file tags and log tag.

- Most paths should be **Fully Qualified Paths** specified from the root of the disk on which Lasso Service is installed. For example, the following path would represent a file in the same folder as Lasso Service in a typical install on a Windows 2000 machine:

```
C://Program Files/Blue World Communications/Lasso Professional 7/default.lasso
```

- The following path would represent the same file if it were in the same folder as Lasso Service in a typical install on a Mac OS X machine:

`///Applications/Lasso Professional 7/default.lasso`

In Mac OS X, the hard drive name is set to a slash / so the fully qualified paths must start with three slashes ///. Paths starting with a single slash / are defined to be relative to the Web server root.

For more information about specifying fully qualified paths, consult *Chapter 20: Files and Logging*.

Note: Fully qualified paths can also be specified in a platform specific fashion. For example, the path above could be written as `C:\Program Files\Blue World Communications\Lasso Professional 7\default.lasso` on Windows or as `Applications:Lasso Professional 7:default.lasso` on Macintosh.

Format File Execution Time Limit

Lasso includes a limit on the length of time that a format file will be allowed to execute. This limit can help prevent errors or crashes caused by infinite loops or other common coding mistakes. If a format file runs for longer than the time limit then it is killed and a critical error is returned and logged.

The execution time limit is set to 10 minutes (600 seconds) by default and can be modified or turned off in the *Setup > Global > Settings* section of Lasso Admin. The execution time limit cannot be set below 60 seconds.

The limit can be overridden on a case by case basis by including the `[Lasso_ExecutionTimeLimit]` tag at the top of a format file. This tag can set the time limit higher or lower for the current page allowing it to exceed the default time limit. Using `[Lasso_ExecutionTimeLimit: 0]` will deactivate the time limit for the current format file altogether.

On servers where the time limit should be strictly enforced, access to the `[Lasso_ExecutionTimeLimit]` tag can be restricted in the *Setup > Global > Tags* and *Security > Groups > Tags* sections of Lasso Admin.

Asynchronous tags and compound expressions are not affected by the execution time limit. These processes run in a separate thread from the main format file execution.

Note: When the execution time limit is exceeded the thread that is processing the current format file will be killed. If there are any outstanding database requests or network connections open there is a potential for some memory to be leaked. The offending page should be reprogrammed to run faster or exempted from the time limit using `[Lasso_ExecutionTimeLimit: 0]`. Restarting Lasso Service will reclaim any lost memory.

4

Chapter 4

LDML 7 Tag Language

This chapter introduces the methodology behind programming data-driven Web sites powered by Lasso 7. This chapter introduces terminology which is used through the remainder of this language guide. All new users of Lasso Professional 7 should read through this chapter to familiarize themselves with the structure of Lasso Dynamic Markup Language (LDML).

- *Introduction* describes the layout of this chapter in detail.
- *Syntax Types* describes the ways to embed LDML 7 tags in format files.
- *Tag Types* introduces the five types of LDML 7 tags including substitution tags, process tags, container tags, member tags, and command tags.
- *Tag Categories and Naming* introduces the logic behind the names of LDML 7 tags.
- *Parameter Types* describes the different types of parameters that can be specified within a tag.
- *Encoding* contains a discussion of character encoding features for substitution tags.
- *Data Types* describes the different data types which LDML 7 offers.
- *Expressions and Symbols* introduces the concept of performing calculations directly within parameters.
- *Delimiters* includes a technical description of the characters used to delimit LDML 7 tags in any syntax.

Introduction

This chapter describes the syntax features of LDML 7. Most of the topics in this chapter are interrelated, and many of the terms used in this chapter are

defined in *Appendix A: Glossary* of the Lasso Professional 7 Setup Guide. Consult this glossary if you are unsure of how any terms are used in this guide.

The first part of this chapters describes the various syntax types that can be used when coding in LDML, and the describes the different categories of LDML tags.

The next part of the chapter describes the syntax of individual tags. The different components of tags are discussed, followed by an introduction to the various parameters that can be specified in LDML tags. Next, the focus shifts to the values which are used to specify parameters. A discussion of Lasso’s built-in data types sets the stage for the introduction of symbols and expressions which can be used to modify values.

Finally, the chapter ends with a technical description of the delimiters used to specify all the different tag types within Lasso and a brief discussion of syntax rules and guidelines which make coding format files within Lasso easier.

Syntax Types

LDML tags can be specified in several different ways within a format file. They can be embedded in square brackets, LassoScripts, compound expressions, HTML form inputs or URLs. Each of these methods is listed in the table below and then described in more detail in the sections that follow:

Table 1: LDML 7 Syntax Types

Syntax Type	Example
Square Brackets	[Field: 'Company_Name']
LassoScript	<?LassoScript Field: 'Company_Name'; ?>
Compound Expression	[Output: {If: \$Num == '1'; Return:'Yes'; /If;}>Run]
HTML Form Inputs	<input type="hidden" name="-Required">
URLs	http://www.example.com/default.lasso?-Token.Num=32

Square Brackets

A single LDML tag can be embedded within square brackets in a format file by specifying its tag name and parameters within the brackets. The entire square bracketed tag will be replaced by the result of the tag when the format file is served to a client. For example, the following [Field] tag is replaced by the value of the specified field in the current database:

[Field: 'Image_URL'] → /Images/Portrait.gif

The square brackets serve to distinguish LDML tags from markup tags in the format file, such as HTML and XML tags which are delimited by angle brackets. LDML tags can be written on their own or within HTML or XML tags. Lasso will not disturb the markup tags, but will replace the square bracketed tag by its value in place when the format file is served. For example, the [Field] tag can be used to specify the value of the src attribute for an HTML tag:

 →

Any of the various tag types can be embedded within square brackets. See the section on *Tag Types* below for more details. Some tags do not return a value in which case they will be evaluated and removed from the output when the format file is served.

Note: Lasso will attempt to interpret any expression that is contained within square brackets in a format file and return the results. See **Chapter 31: Upgrading Your Solutions** for information about how to use square brackets in JavaScript without having Lasso interpret their contents.

LassoScript

Multiple LDML tags can be embedded within a LassoScript in a format file by specifying the tags inside the LassoScript container <?LassoScript ... ?>. The entire LassoScript is replaced by the result of all the tags included in the LassoScript when the format file is served to a client. For example, the following LassoScript will return the value of the included [Field] tag:

```
<?LassoScript
  Field: 'Image_URL';
?>
```

→ /Images/Portrait.gif

Individual LDML tags inside a LassoScript are separated by semi-colons. Multiple tags can be included in the same line as long as they are separated by semi-colons, but usually each tag is specified in its own line. Parentheses can optionally be used around individual tags in order to make it clear which parameters belong to which tag. For example, the following LassoScript contains two [Output] tags and a [Field] tag each specified in its own line. The result of the LassoScript is the concatenation of the values of all three tags.

```
<?LassoScript
  Output: '<img src=\"';
  Field: 'Image_URL';
  Output: '\">';
?>
```

→

The same LassoScript as in the previous example can be written in a single line with optional parentheses included so that the parameters of each tag can be clearly distinguished. Note that even when parentheses are specified around a tag, the semi-colon still must be included between tags:

```
<?LassoScript
  (Output: '<img src=\"'; (Field: 'Image_URL'); (Output: '\">');
?>
```

→

Comments can be included at the end of any line of a LassoScript after two forward slash characters //. The comment continues only until the end of the line. Longer comments can be created by starting subsequent lines with the // characters. For example, the LassoScript from above can be written as follows with comments explaining each of the elements of the LassoScript. The output is the same as above since the comments are all disregarded when the LassoScript is processed:

```
<?LassoScript
  // A LassoScript to output an HTML <img> tag for field 'Image_URL'.
  Output: '<img src=\"';
  // Output the start of the <img> tag up to the first quote mark.
  Field: 'Image_URL';
  // Output the value of the field 'Image_URL' from the database.
  Output: '\">';
  // Output the end of the <img> tag from the final quote mark.
?>
```

→

Any of the various tag types can be embedded within LassoScripts. See the section on *Tag Types* later in this chapter for more details.

Compound Expression Syntax

Compound expression syntax is a combination of square bracket syntax and LassoScript whereby a LassoScript expression can be contained within a custom tag with square bracket syntax. In the example below, a LassoScript conditional statement is used within the [Output] tag to display Yes or No based on the value of the variable MyTest.

```
[Variable: 'myTest'= 'Yes']
[Output: { If: $myTest; Return: 'Yes'; Else; Return: 'No'; /If; }->Eval]
```

→ Yes

Instead of using the LassoScript container `<?LassoScript ... ?>`, the LassoScript syntax is delimited by curly braces `{...}` within square bracket syntax. Inside the curly braces, all syntax rules for LassoScript apply. Compound expression syntax combines the conciseness of LassoScript with the distinguishability of square bracket syntax for streamlined coding.

Compound expression syntax is most useful when creating custom tags, and is described in detail in *Chapter 5: Advanced Programming Topics* of the Extending Lasso 7 Guide. The `[Tag]` tag, which is required for use with compound expressions, is also described in that chapter.

HTML Form Inputs

LDML tags can be embedded within HTML form inputs in two different ways. An LDML command tag can be embedded as the name parameter of an `<input>`, `<select>`, or `<textarea>` tag. LDML tags in square brackets can be embedded as either the name or value parameters. For example, the following `<input>` tag includes an LDML command tag `-ResponseAnyError` as the name parameter and an LDML substitution tag `[Response_FilePath]` as the value parameter.

```
<input type="hidden" name="-ResponseAnyError" value="[Response_FilePath]">
```

When the format file that includes the `-ResponseAnyError` tag is served to a client, the `-ResponseAnyError` tag will not be processed until the HTML form in which this `<input>` is embedded is submitted by a client. However, the `[Response_FilePath]` substitution tag is replaced by the name of the current Web page to yield the following HTML for the `<input>` tag.

→ `<input type="hidden" name="-ResponseAnyError" value="/form.lasso">`

Any of the various tag types can be embedded within HTML form inputs, but the details differ for each type of tag. See the section on *Tag Types* below for more details.

URLs

LDML tags can be embedded within the parameters of URLs in two different ways. An LDML command tag can be embedded as the name half of a parameter. LDML tags in square brackets can be embedded as either the name or value half of a parameter. For example, the following URL includes an LDML command tag `-Token.Name` as the name half of the first

parameter and an LDML substitution tag [Client_Username] as the value half of the first parameter.

```
<a href="http://www.example.com/default.lasso?-Token.Name=[Client_Username]">
```

When the format file that includes this tag is served to a client the -Token.Name command tag will remain unchanged. This tag will not be processed until the URL is selected by a client. The [Client_Username] substitution tag will be replaced by the name of the current user logged in.

➔

```
<a href="http://www.example.com/default.lasso?-Token.Name=Administrator">
```

Any of the various tag types can be embedded within URLs, but the details differ for each type of tag. See the section on *Tag Types* below for more details.

Tag Types

LDML 7 tags are divided into five different types depending on how the tags are used and how their syntax is specified. Each of the five tag types is listed in the table below and then discussed in more detail in the sections that follow, including details of how each tag type can be used within a format file.

Table 2: LDML 7 Tag Types

Tag Type	Example
Substitution Tag	[Field: 'Company_Name']
Process Tag	[Event_Schedule: -URL='http://www.example.com/']
Member Tag	[Output: 'String'->(Get: 3)]
Container Tag	[Loop: 5] ... Looping Text ... [/Loop]
Command Tag	<input type="hidden" name="-Required">

Substitution Tags

Substitution tags return a value which is substituted in place of the tag within the format file being served to a client. Most of the tags in LDML are substitution tags. Substitution tags are used to return field values from a database query, return the results of calculations, or to display information about the state of Lasso Service and the current page request.

The basic format for substitution tags is a tag name followed by a colon and then one or more parameters separated by commas. Every substitution tag also accepts an optional encoding keyword as described later. The

following example shows the structure of substitution tags expressed in square brackets:

```
[Substitution_Tag: Tag_Parameter, -EncodingKeyword]
```

Substitution tags have the same basic form when they are expressed in a LassoScript as when they are expressed in square brackets, except that each tag must end with a semi-colon when expressed in a LassoScript. The following example shows the format of substitution tags and process tags expressed in a LassoScript:

```
<?LassoScript
  Substitution_Tag: Tag_Parameter, -EncodingKeyword;
?>
```

To embed a substitution tag within square brackets:

- Specify the substitution tag on its own. The tag will be replaced by its value when the page is served to a client. For example, the following [Field] tags will be replaced by the company's information from the database:

```
[Field: 'Company_Name'] → Blue World
[Field: 'Company_URL'] → http://www.blueworld.com
```

- Specify the substitution tag within HTML or XML markup tags. The LDML tag will be replaced by its value when the page is served to a client, but the markup tags will be served as written. For example, the following [Field] tags are replaced by the company's information from the database within an HTML anchor tag.

```
<a href="[Field: 'Company_URL']">[Field: 'Company_Name']</a>
```

→ Blue World

To embed a substitution tag within a LassoScript:

- Specify the substitution tag inside the LassoScript container followed by a semi-colon. The value of the LassoScript will be the value of the lone substitution tag. For example, the [Field] tag is the value of the LassoScript in the following code:

```
<?LassoScript
  Field: 'Company_Name';
?>
```

→ Blue World

- Specify multiple substitution tags on separate lines of the LassoScript. End each tag with a semi-colon. The value of the LassoScript will be the concatenation of the value of all the substitution tags. For example,

the [Output] tags and [Field] tag define the value of the LassoScript in the following code:

```
<?LassoScript
  Output: '<b>', -EncodeNone;
  Field: 'Company_Name';
  Output: '</b>', -EncodeNone;
?>
```

→ Blue World

Note: Every substitution tag accepts an optional encoding parameter which specifies the output format for the value which is being returned by the tag. Please see the section on *Encoding* below for more details.

Process Tags

Process tags perform an action which does not return a value. They can be used to alter the HTTP header of an HTML file being served, to store values, to schedule tasks for later execution, to send email messages, and more.

The basic format for process tags is identical to substitution tags: a tag name followed by a colon and then one or more parameters separated by commas.

[Process_Tag: Tag_Parameter]

Process tags have the same basic form when they are expressed in a LassoScript as when they are expressed in square brackets. Except that each tag must end with a semi-colon when expressed in a LassoScript. The following example shows the format of process tags expressed in a LassoScript:

```
<?LassoScript
  Process_Tag: Tag_Parameter;
?>
```

To embed a process tag within square brackets:

- Specify the process tag on its own. The tag will be removed from the format file when it is served. For example, the following [Email_Send] tag will send an email to a specified email address, but will return no value in the Web page being served.

```
[Email_Send: -Host='smtp.myserver.com',
  -To='Somebody@example.com',
  -From='Nobody@example.com',
  -Subject='This is the subject of the email',
  -Body='This is the message text of the email']
```

To embed a process tag within a LassoScript:

- Specify the process tag inside the LassoScript container followed by a semi-colon. Since the process tag does not return a value it will not affect the return value of the LassoScript. For example, the following [Email_Send] tag will send an email to a specified email address, but since the LassoScript contains only this tag it will return no value in the format file being served:

```
<?LassoScript
  Email_Send: -Host='smtp.myserver.com',
              -To='Somebody@example.com',
              -From='Nobody@example.com',
              -Subject='This is the subject of the email',
              -Body='This is the message text of the email';
?>
```

A combination of substitution and process tags can be included in a LassoScript, but the output value of the LassoScript will be determined solely by the value of the substitution tags.

Member Tags

Member tags modify or return data from a value of a specific data type. Each data type in Lasso has different member tags. Member tags can either be used in the fashion of process tags to alter a value or they can be used in the fashion of substitution tags to return a value.

Member tags differ from substitution and process tags in that they must be called using the member symbol -> and a value from the appropriate data type. The following example shows the structure of member tags:

```
[Value->(Tag_Name: Parameters)]
```

For example the [String->Get] member tag requires a value of type string. Member tags are always written in this fashion in the documentation: the data type followed by the member symbol and the specific tag name. The following code fetches the third character of the specified string literal:

```
[Output: 'The String'->(Get: 3)] → e
```

Member tags are defined for any of the built-in data types and third parties can create additional member tags for custom data types. The built-in data types include String, Integer, Decimal, Map, Array, and Pair. More information can be found in the section on *Data Types* below.

To embed a member tag within square brackets:

- Specify the member tag as the parameter of an [Output] substitution tag. This makes it clear that you want to output the value returned by the member tag.

```
[Output: 'The String'-(Get: 3)] → e
[Output: 123-(Type)] → Integer
```

To embed a member tag within a LassoScript:

- Specify the member tag as the parameter of an [Output] substitution tag. This makes it clear that you want to output the value returned by the member tag.

```
<?LassoScript
  Var:'Text'='The String';
  Output: $Text-(Get: 3);
?>
```

→ e

- Member tags can be specified directly if they are being used in the fashion of a process tag. In the following example, the [String->Append] member tag is used to add text to the string, but no result is returned.

```
<?LassoScript
  Var:'Text'='The String';
  $Text-(Append: ' is longer. ');
?>
```

Container Tags

Container tags are a matching pair of tags which enclose a portion of a format file or LassoScript and either alter the enclosed contents or change the behavior of tags within the enclosed contents. The opening tag uses the same syntax as a substitution or process tag. The closing tag has the same name as the opening tag, but the closing tag is specified with a leading forward slash. This is similar to how HTML markup tags are paired.

In the documentation, container tags are referred to by specifying both tags with an ellipsis representing the enclosed content. The loop tag will be referred to as [Loop] ... [/Loop]. When the attributes or parameters of one half of the container tag pair is being discussed, then just the single tag will be named. The opening loop tag is [Loop] and the closing loop tag is [/Loop].

For example, the following [Loop] tag has a single parameter which specifies the number of times the contents of the tag will be repeated. The [/Loop] tag defines the end of the area which will be repeated:

[Loop: 5] Repeated [/Loop]

→ Repeated Repeated Repeated Repeated Repeated

To embed a container tag within square brackets:

- Specify the opening container tag followed by the contents of the container tags and the closing container tag. The contents of the container tags will be affected by the parameters passed to the opening container tag. For example, the following [If] tag will output its contents if its parameter evaluates to True. Since 1 does indeed equal 1 the output is True.

```
[If: 1 == 1] True [/If] → True
```

Note: Both the opening and closing tags of a container tag must be contained within the same format file. Container tags can be nested, but all enclosed container tags must be closed before the enclosing container tag is closed. See *Chapter 13: Conditional Logic* for more information.

To embed a container tag within a LassoScript:

- Specify the opening container tag followed by the contents of the container tag and the closing container tag. Each tag must end with a semi-colon. For readability, the contents of a container tag is often indented. For example, the following [If] tag will output the contents of the enclosed tags if its parameter evaluates to True. Since 1 does indeed equal 1 the output is True.

```
<?LassoScript
  If: 1 == 1;
    Output: True;
  /If;
?>
```

→ True

Command Tags

Most command tags are actually parameters of the [Inline] tag, but can be used on their own within HTML forms or URLs. Command tags are used to send additional information in a form submission or URL request that is flagged for special use by Lasso. This includes specifying field search operators, required form fields, error response pages, and passing token information.

Command tags names always start with a hyphen, e.g. -Required. Command tags can be thought of as “floating parameters”, as they use the same

hyphenated syntax conventions as substitution, process, and container tag parameters, and can also be used directly as [Inline] tag parameters.

The basic format for a command tag is a tag name starting with a hyphen and an associated value. Since command tags can be specified within HTML form inputs, URLs, and as parameters of the [Inline] tag, the form of a command tag is different in each situation.

To embed command tags within an HTML form:

- Specify multiple command tags within the HTML form inputs. Each command tag should be specified in its own form input with the command tag as the name of the input tag.

```
<input type="hidden" name="-CommandTag" value="Command Value">
```

The following example shows a form that contains Lasso command tags. Each -Operator command tag is contained in an HTML hidden input, which augments a field inputs below it. When the form is submitted, each field passed to the searchresponse.lasso page will be passed with an Equals operator, meaning the field value submitted must match values in a database exactly before results will be returned.

```
<form action="searchresponse.lasso" method="post">
  <input type="hidden" name="-Operator" value="equals">
  <input type="text" name="Field1" value="">
  <input type="hidden" name="-Operator" value="equals">
  <input type="text" name="Field2" value="">

  <input type="submit" value="Search">

</form>
```

- Command tags occasionally accept a parameter which is specified just after the name of the tag following a period. For example, the -Token tag has a name parameter and a value parameter. The -Token tag can be specified in a form as follows:

```
<input type="text" name="-Token.Name" value="Default Value">
```

To embed command tags within a URL:

- Specify multiple command tags within the parameters of the URL. A URL consists of a page reference followed by a question mark and one or more URL parameters. Each command tag parameter should be specified as the command tag followed by an equal sign then its value. Individual command tag parameters should be separated in the URL by ampersands.

```
http://www.example.com/default.lasso?-CommandTag=Command%20Value
```

A full action would be specified as follows. The result of selecting this URL in a Web browser would be that the response page `searchresponse.lasso` will be returned to the visitor with the result of the search from the specified database and table.

```
http://www.example.com/searchresponse.lasso?-Operator=Equals&Field1=Value1&
-Operator=Equals&Field2=Value2
```

To embed command tags within an `[Inline]`:

- Specify multiple command tags within the opening `[Inline]` tag. The command tags will specify the action which the `[Inline]` is to perform. The contents of the `[Inline] ... [/Inline]` tags will be affected by the results of this action. The following example shows how the `-Op` tags can be used directly within an `[Inline]` tag.

```
[Inline:
  -Database='Contacts'
  -Table='People',
  -KeyField='ID',
  -Op='eq',
  'Field1'='Value1',
  -Op='eq',
  'Field2'='Value2',
  -Search]
...
[/Inline]
```

Tag Categories and Naming

All of the tags in LDML 7 are grouped and named according to a few simple rules. These rules define where the tag can be found in Lasso 7 documentation and in Lasso Administration.

Tag Categories

The following chart describes the major tag categories in LDML 7. Each tag category is discussed in more detail later in the book. Look for a chapter which has the same name as the tag category or use the index to locate a particular tag.

Table 3: LDML 7 Tag Categories

Tag Category	Description
Administration	Administration and security tags.
Array	Array, map, and pair member tags.

Client	Information about the current visiting client.
Conditional	Conditional logic and looping tags.
Custom Tag	Create custom LDML tags and data types.
Data Types	Tags to cast values to specific data types.
Database	Information about the current database.
Date	Date manipulation tags.
Encoding	Tags for encoding data.
Encryption	Encrypt data so it can be transmitted securely.
Error	Tags for reporting and handling errors.
File	Tags for manipulating files.
Image	Tags for manipulating images.
Include	Allows data to be included in a format file.
Link	Link to other records in the current found set.
Math	Mathematical operations and integer member tags.
Operator	Set and retrieve logical and field-level operators.
Output	Tags for formatting or suppressing output.
Network	Tags for performing network operations.
PDF	Tags for creating PDF documents.
Results	Results from the current Lasso action.
String	String operations and string member tags.
Technical	Tags for performing low-level operations.
Utility	Tags which don't fit in any other category.
Variable	Tags for creating and manipulating variables.
XML	Tags for processing XML.

Tag Naming Conventions

Tags in LDML are named according to a set of well-defined naming conventions. Understanding these conventions will make it easier to locate the documentation for specific tags. We also recommend the following naming conventions when creating custom tags, libraries, and modules.

- Case is unimportant in both tag name and tag parameter names. All LDML tags can be written in uppercase, lowercase, or any combination of mixed case. Tags are always written in title case in the documentation. The following tag names would all be equivalent, but the first, e.g. title case, is preferred:

[Tag_Name]	[tag_name]
[TAG_NAME]	[TaG_NaMe]

- Core language tags usually have simple tag names and do not contain underscore characters. For example:

[Variable]	[Field]
[If] ... [Else] ... [/If]	[Inline] ... [/Inline]

- Most tag names include a category name followed by an underscore then the specific tag name. For example: [Math_Sin] is the tag in the “Math” category that performs the function “Sine.” Similarly, [Link_NextRecordURL] is the tag in the “Link” category that returns the URL of the next record in the found set. Category names appear in tag names based on the following format:

[Category_TagName]

- Tag names never start with an underscore character. These tag names are reserved for internal use.
- Some tag names reference another tag or other component of Lasso 7 followed by an underscore then a specific tag name. For example [MaxRecords_Value] returns the value of the -MaxRecords command tag. There is no underscore in the words MaxRecords since it is referring to another tag. This association can be expressed as follows:

[TagReference_TagName]

- Many tag names include a word at the end that specifies what the output of the tag will be. For instance, [Link_NextRecord] ... [/Link_NextRecord] is a container tag that links to the next record, but [Link_NextRecordURL] is a substitution tag that returns the URL of the next record. Tags that end in “URL” output URLs. Tags that end in “List” and most tags that have plural names output arrays. Tags that end in “Name” return the name of a database entity. Tags that end in “Value” return the value of the named database entity.

[Link_NextRecordURL]	[File_ListDirectory]
[Action_Params]	[Variables]
[KeyField_Name]	[KeyField_Value]

- Member tag names are written in the documentation with the data type followed by the member symbol then the tag name. For example, the Get tag of the data type string would be written: [String->Get]. All of the member tags of a particular data type are considered to be part of the category which has the same name as the data type. All of the string member tags are part of the string category.
- Tags created by third parties should start with a prefix which identifies the creator of the tag. For example, tags from “Example Company” might all start with Ex_. This ensures that the third party tags do not conflict with built-in tags or other third party tags.

[Ex_TagName] [ExCategory_TagName]

Synonyms and Abbreviations

The following charts detail some standard synonyms and abbreviations in LDML 7. Any of the synonyms or abbreviations in the right column can be used instead of the term in the left column, but the term in the left column is preferred.

Table 4: LDML 7 Synonyms

Preferred Term	Synonym	Example
Field	Column	[Field_Name] [Column_Name]
Record	Row	[Records] [Rows]
KeyValue	RecordID	[KeyField_Value] [RecordID_Value]
Table	Layout	[Table_Name] [Layout_Name]

Table 5: LDML 7 Abbreviations

Preferred Term	Abbreviation	Example
Operator	Op	-Operator -Op
Required	Req	-Required -Req
Variable	Var	[Variable] [Var]

Some tags which were synonyms in earlier version of Lasso are no longer supported. Please see *Chapter 31: Upgrading Your Solutions* for more information. For a complete list of synonyms and abbreviations please consult the LDML 7 Reference.

Parameter Types

This section introduces the different types of parameters which can be specified within LDML tags. This discussion is applicable to substitution tags, process tags, the opening tag of container tags, and member tags. Command tag parameters are fully described in the previous section.

Table 6: Parameter Types

Parameter Type	Example
Value	[Field: 'Field_Name']
Keyword	[Error_CurrentError: -ErrorCode]
Keyword/Value	[Inline: -Database=(Database_Name), ...]
Name/Value	[Variable: 'Variable_Name'='Variable_Value']

Some parameters are required for a tag to function properly. The [Field] and [Variable] tags require that the field or variable to be returned is specified. In contrast, the keyword in [Error_CurrentError] is optional and can be safely omitted. If no keyword is specified for an optional parameter then a default will be used. For a complete listing of required, optional, and default parameters for each tag, please consult the LDML 7 Reference.

A **Value** is the most basic parameter type, and consists of a basic data type contained within a tag after a colon character (:). Values include string literals, integer literals, decimal literals, sub-tags, and complex expressions.

[Field: 'Field_Name']	[Date: '09/29/2003']
[Var_Defined: 'Variable_Name']	[Output: 123]

A value can also be the value of a sub-tag. Any substitution tag or member tag can be used as a sub-tag. The syntax of the sub-tag is the same as that for the substitution tag or member tag except that the tag is enclosed in parentheses rather than square brackets. The following [Output] tags are used to output the value of several different sub-tags:

[Output: (Field: 'Field_Name')]	[Output: (Date)]
[Output: (Loop_Count)]	[Output: 'String'->(Get: 3)]

A **Keyword** is a tag-specific parameter that alters the behavior of a tag. Keyword names always start with a hyphen. This makes it easy to distinguish tag-specific keywords from user-defined parameters. The following examples of [Server_Date] show how the same tag can be used to generate different content based on the keyword that is specified:

```
[Server_Date: -Short] → 3/24/2001
[Server_Date: -Long] → March 24, 2001
[Server_Date: -Abbrev] → Mar 24, 2001
[Server_Date: -Extended] → 2001-03-24
```

Note: For backwards compatibility, some tags will accept keyword names without the leading hyphen. This support is not guaranteed to be in future versions of Lasso so it is recommended that you write all keyword names with the leading hyphen.

A **Keyword/Value** parameter is the combination of a tag specific keyword and a user-defined value which affects the output of a tag. The keyword name is specified followed by an equal sign and the value. Keyword/value parameters are sometimes referred to as named parameters. For example, the [Date] tag accepts multiple keyword/value parameters which specify the characteristics of the date which should be output:

```
[Date: -Year=2001, -Day=24, -Month=3] → 3/24/2001
```

Command tags are used like keyword/value parameters in the [Inline] tag. The command tag functions like the keyword and is written with a leading hyphen. For example, the following [Inline] contains several command tags that define a database action:

```
[Inline: -FindAll
        -Database='Contacts',
        -Table='People',
        -KeyField='ID']
... Results ...
[/Inline]
```

A **Name/Value** parameter is the combination of a user-defined name with a user-defined value. The name and the value are separated by an equal sign. Name/value parameters are most commonly used in the [Inline] tag to refine the definition of a database action. For example, the previous [Inline] example can be modified to search for records where the field First_Name starts with the letter s by the addition of a name/value parameter 'First_Name'='s':

```
[Inline: -Search,
        'First_Name'='s',
        -Database='Contacts',
        -Table='People',
        -KeyField='ID']
... Results ...
[/Inline]
```

Encoding

Encoding keyword parameters specify the character format in which the data output from a substitution tag should be rendered. Encoding ensures that reserved or illegal characters are changed to entities so that they will

display properly in the desired output format. Encoding keywords allow substitution tags to be used to output data in any of the following formats:

- HTML text for display in a Web browser (default).
- HTML tags for display in a Web browser.
- XML data for data interchange.
- URL parameters to construct a hyperlink.
- ASCII text for inclusion in an email message or log file.

The following table demonstrates each of the encoding keywords available in LDML 7.

Table 7: Encoding Keywords

Keyword	Encoding Performed
-EncodeNone	No encoding is performed.
-EncodeHTML	Reserved, illegal, and extended ASCII characters are changed to their hexadecimal equivalent HTML entities.
-EncodeSmart	Illegal and extended ASCII characters are changed to their hexadecimal equivalent HTML entities. Reserved HTML characters are not changed.
-EncodeBreak	ASCII carriage return characters are changed to HTML .
-EncodeURL	Illegal and extended ASCII characters are changed to their equivalent hexadecimal HTTP URL entities.
-EncodeStrictURL	Reserved, illegal and extended ASCII characters are changed to their equivalent hexadecimal HTTP URL entities.
-EncodeXML	Reserved, illegal, and extended ASCII characters are changed to their UTF-8 equivalent XML entities.

To use an encoding keyword:

Append the desired encoding keyword at the end of a substitution tag. For example, angle brackets are reserved characters in HTML. If you want to include an angle bracket in your HTML output it needs to be changed into an HTML entity. The entity for < is < and the entity for > is >.

[Output: 'HTML Text', -EncodeHTML] → HTML Text

See *Chapter 18: Encoding* for more information.

Data Types

Every value in Lasso is defined as belonging to a specific data type. The data type determines what member tags are available and how symbols affect the value. Data types generally correspond to everyday descriptions of a value with the addition of some data types for structured data. The following table lists the primary data types available in Lasso:

Table 8: Primary LDML 7 Data Types

Data Type	Example
String	'This is a string surrounded by single quotes'
Integer	1500
Decimal	3.14159
Date	9/29/2002 19:12:02
Duration	168:00:00
Array	[Array: 'red', 'green', 'blue', 'yellow']
Map	[Map: 'Company_Name'='Blue World', 'City'='Bellevue']

Note: This section describes the primary data types which are used most frequently in LDML. There are many other special-purpose data types in LDML, including **PDF**, **Image**, **File**, and **Network** Types. These special-purpose types are described in appropriate chapters later in this guide.

Strings

Strings are any series of alphanumeric characters. String literals are surrounded by single quotes. The results of a substitution tag will be considered a string if it contains any characters other than numbers. Please see *Chapter 14: String Operations* for more information.

Some examples of string values include:

- 'String literal' is a string surrounded by single quotes.
- '123456' is a string literal since it is surrounded by single quotes.
- 'A string with \"quotes\" escaped' is a string that contains quote marks. The quote marks are considered part of the string since they are preceded by back slashes.
- The following [Field] tag returns a string value. Notice that the value of a substitution tag is a string value since it contains alphabetic characters:
[Field: 'Company_Name'] → Blue World

- The following code sets a variable to a string value, then retrieves that value:

```
[Variable: 'String' = 'abcdef']  
[Variable: 'String'] → abcdef
```

Integers

Integers are any series of numeric characters that represent a whole number. Integer literals are never surrounded by quotes. The results of a substitution tag will be considered an integer if it contains only numeric characters which represent a whole number. Please see *Chapter 15: Math Operations* for more information.

Some examples of integer values include:

- 123456 is an integer literal since it is not surrounded by quotes.
- (-50) is an integer literal. The minus sign (hyphen) is used to define a negative integer literal. The parentheses are required if the literal is to be used as the right-hand parameter of a symbol.
- The following [Field] tag returns an integer value. The value is recognized as an integer since it contains only numeric characters and represents a whole number:

```
[Field: 'Employee_Age'] → 23
```

- The following code sets a variable to an integer value, then retrieves that value:

```
[Variable: 'Integer' = 1000]  
[Variable: 'Integer'] → 1000
```

Decimals

Decimals are any series of characters that represent a decimal number. Decimal literals are never surrounded by quotes. Decimal values must include a decimal point and can be expressed in exponential notation. Please see *Chapter 15: Math Operations* for more information.

Some examples of decimal values include:

- 123.456 is a decimal literal since it contains a decimal point and is not surrounded by quotes.
- (-50.0) is a negative decimal literal. The parentheses are required if the literal is to be used as the right-hand parameter of a symbol.
- The following [Field] tag returns a decimal value. The value is recognized as a decimal since it contains numeric characters and a decimal point:

```
[Field: 'Annual_Percentage_Rate'] → 0.12
```

- The following code sets a variable to a decimal value, then retrieves that value:

```
[Variable: 'Decimal' = 137.48]
[Variable: 'Decimal'] → 137.48
```

Dates

Dates are a special data type that represent a date and/or time string. Dates in Lasso 7 can be manipulated in a similar manner as integers, and calculations can be performed to determine date differences, durations, and more. For Lasso to recognize a string as a date data type, the string must be explicitly cast as a date data type using the [Date] tag. When casting as a date data type, the following date formats are automatically recognized as valid date strings by Lasso:

```
1/1/2001
1/1/2001 12:34
1/1/2001 12:34:56
1/1/2001 12:34:56 GMT
2001-01-01
2001-01-01 12:34:56
2001-01-01 12:34:56 GMT
```

The “/”, “-”, and “.” characters are the only punctuation marks recognized in valid date strings by Lasso. If using a date format not listed above, custom date formats can be defined as date data types using the [Date] tag with the -Format parameter. See *Chapter 16: Date and Time Operations* for more information.

Some examples of dates include:

- [Date:'9/29/2002'] is a valid date data type recognized by Lasso.
- [Date:'9.29.2002'] is not recognized by Lasso as a valid date data type due to its punctuation, but can be converted to a date data type using the [Date] tag with the -Format parameter.

```
[Date:'9.29.2002', -Format='%m.%d.%Y']
```

- Specific date and time information can be obtained from date data types using accessors.

```
[(Date:'9/29/2002')->DayofYear] → 272
```

- Date data types can be manipulated using math symbols. Date and time durations can be specified using the [Duration] tag.

```
[(Date:'9/29/2002') + (Duration: -Day=2)] → 10/01/2002
```

- A valid date data type can be displayed in an alternate format using the [Date_Format] tag.

[Date_Format:(Date:'9/29/2002'), -Format='%Y-%m-%d'] → 2002-09-29

Note: Lasso uses internal standardized date libraries to automatically adjust for leap years and day light savings time when performing date calculations. The current time and time zone are based on that of the Web server. For information on special cases with date calculations during day light saving time, see *Chapter 16: Date and Time Operations*.

Durations

Durations are a special data type that represent a length of time in hours, minutes, and seconds. Durations are not 24-hour clock times, and can represent any length of time. Duration data types in Lasso 7 are related to date data types, and can be manipulated in a similar manner. For Lasso to recognize a string as a duration data type, the string must be explicitly cast as a duration data type using the [Duration] tag. Any numeric string formatted as hours:minutes:seconds or just seconds may be cast as a duration data type.

```
168:00:00
60
```

Colon characters (:) are the only punctuation marks recognized in valid duration strings by Lasso. The [Duration] tag always outputs values in hours:minutes:seconds format regardless of what the input format was. See *Chapter 16: Date and Time Operations* for more information.

Some examples of durations include:

- [Duration:'169:00:00'] is a valid duration data type recognized by Lasso, and represents a duration of 169 hours. This duration will be output as 169:00:00.
- [Duration:'300'] is a valid duration data type recognized by Lasso, and represents a duration of 300 seconds. This duration will be output as 00:05:00 (five minutes).

Arrays

Arrays are a series of values which can be stored and retrieved by numeric index. Arrays can contain values of any other data type, including other arrays. Only certain substitution tags return array values. Array values are never returned from database fields. Please see *Chapter 17: Arrays and Maps* for more information.

Some examples of how to work with arrays include:

- Create an array using the [Array] tag. The following two examples create an array with the days of the week in it, where each day of the week is a

string literal. The second example shows abbreviated syntax where the colon (:) character is used to specify the start of an array data type.

```
[Array: 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
[: 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

- Store an array in a variable using the following code which stores the array created in the code above in a variable named Week.

```
[Variable: 'Week' = (Array: 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday', 'Sunday')]
```

- Fetch a specific item from the array using the [Array->Get] member tag. This code fetches the name of the third day of the week.

```
[Output: (Variable: 'Week')->(Get:3)] → Wednesday
```

- Set a specific item from the array using the [Array->Get] member tag. The following code sets the name of the third day of the week to its Spanish equivalent Miercoles.

```
[(Variable: 'Week')->(Get:3) = 'Miercoles']
```

The new value of the third entry in the array can now be fetched.

```
[Output: (Variable: 'Week')->(Get:3)] → Miercoles
```

Maps

Maps are a series of values which can be stored and retrieved by name. Maps can contain values of any other data type, including arrays or other maps. Only certain substitution tags return map values. Map values are never returned from database fields. Please see *Chapter 17: Arrays and Maps* for more information.

Some examples of how to work with maps include:

- Create a map using the [Map] tag. The following creates a map with some user information in it. The name of each item is a string literal, the values are either string literals or decimal literals:

```
[Map:
    'First Name'='John',
    'Last Name'='Doe',
    'Age'=25]
```

- Store a map in a variable using the following code which stores the map created in the code above in a variable named Visitor:

```
[Variable: 'Visitor' = (Map:
    'First Name'='John',
    'Last Name'='Doe',
    'Age'=25)]
```

- Fetch a specific item from the map using the [Map->Get] member tag. This code fetches the visitor's first name:

```
[Output: (Variable: 'Visitor')->(Get:'First Name')] → John
```

- Set a specific item from the map using the [Map->Get] member tag. This code sets the age of the visitor to 29. Notice that the [Output] tag returns no value since the member tag is being used in the fashion of a process tag to set a value.

```
[Output: (Variable: 'Visitor')->(Get:'Age') = 29]
```

The new value of the age entry in the map can now be fetched:

```
[Output: (Variable: 'Visitor')->(Get:'Age')] → 29
```

Note: There are other, less common data types in LDML that are not defined here. These include pair, boolean, and null. Please see *Chapter 17: Arrays and Maps* for more information about the pair type, *Chapter 22: Control Tags* for more information about the null type, and *Chapter 13: Conditional Logic* for more information about the boolean type.

Expressions and Symbols

Virtually all of the values shown in this chapter so far have been simple string, integer or decimal literals. Any tag in LDML which accepts a value as a parameter can accept an expression in place of that value. This allows nested operations to be performed within the parameters of LDML tags.

This section discusses each of the different types of expressions that can be used as values within LDML. It starts with simple expressions and then moves on to more complex expressions. The [Output] tag will be used throughout this section to output the value of expressions.

Table 9: Types of LDML 7 Expressions

Expression	Example
Literal	'String Literal', 100, 150.34
Sub-Tag	(Variable: 'Variable_Name')
Member tag	(Array: 1, 2, 3, 4)->(Get: 4)
String Expression	'String One' + 'String Two'
Math Expression	100 / 4 + 25 - (-20)
Complex Expression	'' + 100 / 4 + ''
Conditional Expression	'azure' == 'blue'
Logical Expression	('blue' != 'orange') ('red' != 'green')

This section also describes each of the different symbols that can be used to modify expressions specific to each type of expression.

Literals

Any string literal, integer literal, or decimal literal can be used as a value in LDML. These are the most basic types of values and the simplest examples of expressions. These literals are defined in the previous section on *Data Types*. Some examples of outputting literal values include:

[Output: 'String Literal']	[Output: 123]
[Output: 100.14]	[Output: (-123)]

Note: The [Output] tag is not technically required in these expressions. [123] will evaluate to the integer value 123. However, for clarity, the use of the [Output] tag is recommended for displaying expressions.

Sub-Tags

Substitution tags are LDML tags that return a value and any substitution tag can be used as a simple expression in LDML. The syntax of the sub-tag is the same as that for the substitution tag except that the tag is enclosed in parentheses rather than square brackets. The value of the expression is simply the value of the substitution tag. For example, the following [Output] tags output the value of the specified sub-tag.

[Output: (Field: 'Field_Name')]	[Output: (Date)]
[Output: (Loop_Count)]	

Note: Substitution tags have a default encoding keyword of -EncodeHTML applied when they are the outermost tag. However, when substitution tags are used as sub-tags or in square brackets without an [Output] tag, no encoding is applied by default. See *Chapter 18: Encoding* for more information.

Member Tags

Member tags that return values can be used as simple expressions in LDML. An appropriate member tag for any given data type can be attached to a value of that data type using the member symbol ->. For example, the following member tag returns a character from the specified string literal:

[Output: 'String'->(Get: 3)]

The value on the left side of the member symbol can be any expression which is valid in LDML. It can be a string literal, integer literal, decimal literal, sub-tag, or any of the expressions which are defined below. For

example, the following member tag would return the third character of the name which is returned from the database:

```
[Output: (Field: 'First Name')->(Get: 3)]
```

Note: The [Output] tag is not technically required in member tag expressions. [String'->(Get: 3)] will evaluate to the character r. However, for clarity, the use of the [Output] tag is recommended.

Table 10: Member Tag Symbol

Symbol	Name	Example
->	Member	[Output: 'abcdef'->(Get: 3)] → c

String Expressions

String expressions are the combination of string values with one or more string symbols. A string expression defines a series of operations that should be performed on the string values. The string values which are to be operated upon can be either string literals or any expressions which return a string value.

Symbols should always be separated from their parameters by spaces and string literals should always be surrounded by single quotes. Otherwise, Lasso may have a difficult time distinguishing literals and LDML tags.

The most common string symbol is + for concatenation. This symbol can be used to combine multiple string values into a single string value. For example, to add bold tags to the output of a [Field] tag we could use the following string expression:

```
[Output: -EncodeNone, '<b>' + (Field: 'CompanyName') + '</b>']
```

```
→ <b>Blue World</b>
```

String symbols can also be used to compare strings. String symbols can check if two strings are equal using the equality == symbol or can check whether strings come before or after each other in alphabetical order using the greater than < or less than > symbols. For example, the following code reports the proper order for two strings:

```
[If: 'abc' == 'def']
  abc equals def
[Else: 'abc' < 'def']
  abc comes before def
[Else: 'abc' > 'def']
  abc comes after def
[/If]
```

```
→ abc comes before def
```

Note: Always place spaces between a symbol and its parameters. The - symbol can be mistaken for the start of a negative number, command tag, keyword, or keyword/value parameter if it is placed adjacent to the parameter that follows.

Table 11: String Expression Symbols

Symbol	Name	Example
+	Concatenation	[Output: 'abc' + 'def'] → abcdef
*	Repetition	[Output: 'abc' * 2] → abcabc
-	Deletion	[Output: 'abcdef' - 'cde'] → abf
>>	Contains	[Output: 'abcdef' >> 'bcd'] → True
!>>	Not Contains	[Output: 'abcdef' !>> 'bcd'] → False
==	Equality	[Output: 'abc' == 'def'] → False
!=	Inequality	[Output: 'abc' != 'def'] → True
<	Less Than	[Output: 'abc' < 'def'] → True
>	Greater Than	[Output: 'abc' > 'def'] → False

Please see *Chapter 14: String Operations* for more information.

Math Expressions

Math expressions are the combination of decimal or integer values with one or more math symbols. A math expression defines a series of operations that should be performed on the decimal or integer values. The numeric values which are to be operated upon can be either decimal or integer literals or any expressions which return a numeric value.

Symbols should always be separated from their parameters by spaces. This ensures that the + and - symbols are not mistaken for the sign of one of the parameters.

Simple math operations can be performed directly within an expression. For example, the following [Output] tags return the value of the specified simple math calculations.

[Output: 10 + 5] → 15	[Output: 10 - 5] → 5
[Output: 10 * 5] → 50	[Output: 10 / 5] → 2

If the second parameter of the expression is negative it should be surrounded by parentheses.

[Output: 10 + (-5)] → 5	[Output: 10 * (-5)] → -50
-------------------------	---------------------------

Math expressions can be used on either decimal or integer values. If both parameters of a math symbol are integer values then an integer result will be returned. However, if either parameter of a math symbol is a decimal

value then a decimal value will be returned. Decimal return values always have at least six significant digits.

Note: Always place spaces between a symbol and its parameters. The - symbol can be mistaken for the start of a negative number, command tag, keyword, or keyword/value parameter if it is placed adjacent to the parameter that follows.

Table 12: Math Expression Symbols

Symbol	Name	Example
+	Addition	[Output: 100 + 25] → 125
-	Subtraction	[Output: 100 - 25] → 75
*	Multiplication	[Output: 100 * 25] → 2500
/	Division	[Output: 100 / 25] → 4
%	Modulo	[Output: 100 % 25] → 0
==	Equality (Value Only)	[Output: 100 == 25] → False
===	Equality (Value & Type)	[Output: 100 == 100.0] → False
!=	Inequality (Value Only)	[Output: 100 != 25] → True
>	Greater Than	[Output: 100 > 25] → True
>=	Greater Than or Equal	[Output: 100 >= 25] → True
<	Less Than	[Output: 100 < 25] → False
<=	Less Than or Equal	[Output: 100 <= 25] → False

Please see *Chapter 15: Math Operations* for more information.

Complex Expressions

Complex expressions can be created by combining sub-expressions together using one or more string or math symbols. The results of the sub-expressions are used as the parameters of the enclosing parameters. Expressions can be enclosed in parentheses so that the order of operation is clear.

For example, the following complex math expression contains many nested math expressions. The expressions in the innermost parentheses are processed first and the result is used as a parameter for the enclosing expression. Notice that spaces are used on either side of each of the mathematical symbols.

[Output: (1 + (2 * 3) + (4.0 / 5) + (-6))] → 1.8

The following complex string expressions contains many nested string expressions. The expressions in the innermost parentheses are processed first and the result is used as a parameter for the enclosing expression:

[Output: ('abc' + ('def' * 2) + ('abcdef' - 'def') + 'def')] → abcdefdefabcdef

String and math expressions can be combined. The behavior of the symbols in the expression is determined by the parameters of the symbol. If either parameter is a string value then the symbol is treated as a string symbol. Only if both parameters are decimal or integer values will the symbol be treated as a math symbol. For example, the following code adds two numbers together using the math addition + symbol and then appends bold tags to the start and end of that value using the string concatenation + symbol:

[Output: '' + (100 + (-35)) + '', -EncodeNone] → 65

Conditional Expressions

Conditional expressions are the combination of values of any data type with one or more conditional symbols. A conditional expression defines a series of comparisons that should be performed on the parameter values. The values which are to be operated upon can be valid values or expressions.

Conditional symbols were introduced in the *String Expressions* and *Math Expressions* sections above in the context of comparing string or math values. They can actually be used on values of any data type including arrays, maps, and custom types defined by third parties.

Values are automatically converted to an appropriate data type for a comparison. For example, the following comparison returns `True` even though the first parameter is a number and the second parameter is a string. The second parameter is converted to the same type as the first parameter, then the values are compared:

[Output: 123 == '123'] → True

Conditional expressions are used in the `[If] ... [/If]` and `[While] ... [/While]` container tags to specify the condition under which the contents of the tag will be output. For example, the following `[If]` tag contains a conditional expression that will evaluate to `True` only if the company name is Blue World:

```
[If: (Field: 'Company_Name') == 'Blue World']
  The company name is Blue World
[/If]
```


Table 13: Conditional Expression Symbols

Symbol	Name	Example
>>	Contains	[Output: 'abcdef' >> 'bcd'] → True
!>>	Not Contains	[Output: 'abcdef' !>> 'bcd'] → False
==	Equality (Value Only)	[Output: 100 == 25] → False
===	Equality (Value & Type)	[Output: 100 == '100'] → False
!=	Inequality (Value Only)	[Output: 100 != 25] → True
>	Greater Than	[Output: 100 > 25] → True
>=	Greater Than or Equal	[Output: 100 >= 25] → True
<	Less Than	[Output: 100 < 25] → False
<=	Less Than or Equal	[Output: 100 <= 25] → False

Please see *Chapter 13: Conditional Logic* for more information.

Note: If the second parameter to a conditional expression uses a single-parameter symbol then it should be surrounded by parentheses. For example, `($Variable == (-1))` or `($Variable == (!False))`.

Logical Expressions

Logical expressions are made up of multiple conditional sub-expressions combined with one or more logical symbols. The values of the conditional sub-expressions are combined according to the operation defined by the logical symbol.

Logical expressions are most commonly used in the `[If] ... [/If]` container tag to specify the condition under which the contents of the tag will be output. A single `[If]` tag can check multiple conditional expressions if they are combined into a single logical expressions.

For example, the following `[If]` tag contains a logical expression that will evaluate to `True` if one or the other of the sub-expressions is `True`. The `[If] ... [/If]` container tag will display its contents only if the company name is Blue World or the product name is Lasso Professional:

```
[If: ((Field: 'Company_Name') == 'Blue World') ||
      ((Field: 'Product_Name') == 'Lasso Professional')]
  The company name is Blue World
[/If]
```

Table 14: Logical Expression Symbols

Symbol	Name	Example
&&	And	[Output: True && False] → False
	Or	[Output: True False] → True
!	Not	[Output: ! True] → False

Please see *Chapter 13: Conditional Logic* for more information.

Note: These logical symbols should not be confused with the logical search operators which can be used to assemble complex search criteria. See *Chapter 6: Database Interaction Fundamentals* for more information about logical search operators.

Delimiters

This section describes the delimiters which are used to define LDML and HTML. It is important to understand how delimiters are used so that tags can be constructed with the proper syntax.

Table 15: LDML 7 Delimiters

Symbol	Name	Function
[Square Bracket	Start of tag square bracket syntax.
]	Square Bracket	End of tag in square bracket syntax.
/	Forward Slash	Closing container tag name.
:	Colon	Separates tag name from tag parameters.
,	Comma	Separates tag parameters.
=	Equal Sign	Separates name/value parameter.
-	Hyphen	Starts command tag name and keyword names.
'	Single Quote	Start and end of LDML string value.
(Parentheses	Start of sub-tag or expression.
)	Parentheses	End of sub-tag or expression.
<?LassoScript	LassoScript	Start of LassoScript.
?>	LassoScript	End of LassoScript.
{	Curly Brace	Start of compound expression syntax (LassoScript contained within square bracket syntax).
}	Curly Brace	End of compound expression syntax.
;	Semi-Colon	Separates tags within LassoScript.
//	Double Slash	Start of line comment in LassoScript.
/*	Asterisk Slash	Start of extended comment in LassoScript.
*/	Asterisk Slash	End of extended comment in LassoScript.
->	Member Symbol	Separates data value from member tag.
	Space	Specified between symbols and their parameters.

When possible, parentheses should be used around all expressions, sub-tag calls, and negative literals. The parentheses will ensure that Lasso accurately parses each expression. If an expression does not seem to be working correctly, try adding parentheses to make the order of operation explicit.

Unlike symbols, white space is generally not required around delimiters. White space may be used to format code in order to make it more readable.

Note: The double quote " was a valid LDML separator in earlier versions of Lasso but has been deprecated in Lasso Professional 7. It is not guaranteed to work in future versions of Lasso.

The following table shows the delimiters which are used in HTML pages and HTTP URLs.

Table 16: HTML/HTTP Delimiters

Symbol	Name	Function
<	Angle Bracket	Start of an HTML or XML tag.
>	Angle Bracket	End of an HTML or XML tag.
=	Equal Sign	Separates name/value parameter or attribute.
"	Double Quote	Start and end of HTML string value.
?	Question Mark	Separates path from parameters in URL.
#	Hash Mark	Separates path from target in URL.
&	Ampersand	Separates URL parameters.
/	Forward Slash	Folder delimiter in URL paths or designation of Web server root if used at the start of a URL path.
../	Dot Dot Slash	Up one folder level in URL paths.
	Space	Separates tag attributes.

Illegal Characters

The following chart details characters which can cause Lasso problems if they appear in a format file or within LDML code outside of a string literal. These characters are not valid in tag names, keyword names, or parameter names.

For best results use a dedicated HTML editor such as Macromedia Dreamweaver or Adobe GoLive or a text editor such as BareBones BBEdit or Microsoft NotePad to create LDML format files. The Zap Gremlins option

in BBEEdit is particularly useful in eliminating problem characters such as these.

Table 17: Illegal Characters

Symbol	Name	Function
	Non-Breaking Space	Non-breaking spaces can be used within string literals, but are not valid white space within LDML code. Often typed Option-Space on Macintosh.
\0	Null Character	The null-character is often used as an end-of-file marker. Lasso may abort processing if it reads a null character within a format file.

5

Chapter 5

LDML 7 Reference

This chapter documents how to use the LDML 7 Reference.

- **Overview** provides an overview of the LDML 7 Reference and how to access it.
- **Search** discusses searching the LDML 7 Reference.
- **Browse** discusses browsing the LDML 7 Reference by tag type or category.
- **Detail** discusses how to view information about LDML tags, and what information can be displayed.
- **List** discusses how all available tags can be listed.

Overview

The LDML 7 Reference is a resource provided by Blue World for finding descriptions, usage guidelines, and detailed examples of LDML tags. It is the official reference for all tags in LDML 7.

The LDML 7 Reference is a locally-stored LassoApp and Lasso MySQL database included with each installation of Lasso Professional 7, and is also available on the Blue World Web site.

To access the LDML 7 Reference:

- The LDML 7 Reference can be accessed through the **Support > LDML Reference** section in Lasso Administration.
- The LDML 7 Reference can also be accessed on the local machine at the following URL, substituting the actual IP address or host name of

the Web server for `www.example.com`. The LDML 7 Reference requires the administrator username and password for local access.

`http://www.example.com/Lasso/LDMLReference.LassoApp`

- The LDML 7 Reference can be accessed at Blue World at the following URL. This reference is open for anyone to use and includes a public comment interface.

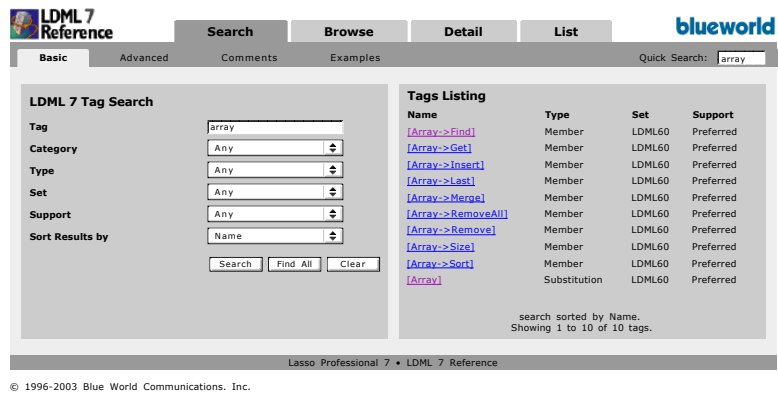
`http://ldml.blueworld.com/`

This version contains the same information as the locally-stored LDML 7 Reference, however, it also contains documentation comments and code examples from users and developers. This is useful for finding further examples and information about particular tags.

Components

The local version of the LDML 7 Reference consists of two components. The interface is provided by the `LDMLReference.LassoApp` file located in the Lasso directory of the Web server root. The data for the reference is stored within Lasso MySQL in a database named `LDML7_Reference`. Both components are installed as part of the standard Lasso Professional 7 installation.

Figure 1: LDML 7 Reference



Sections of the Interface

The interface is divided into four sections, navigable via tabs at the top of the screen. These sections are:

- **Search** – Allows searching the LDML 7 Reference database.

- *Browse* – Allows browsing the LDML 7 Reference database by category.
- *Detail* – Shows descriptions, comments, and examples of specific LDML tags.
- *List* – Shows a listing of all available LDML tags summarized by category.

Navigation

Navigation occurs by selecting the tab for the desired section at the top of the interface. Doing so will display the default screen for that tab and additional tabs for any subsections. Many screens contain two panels. The left panel generally provides a search interface or a list of options. The right panel provides search results or details for any selected option.

Navigation within extended lists occurs via *Prev* and *Next* buttons. Listings are displayed in groups of ten or fifteen depending on the section.

Search

This section describes searching for LDML tags in the LDML 7 Reference using the Search section of the interface.

Basic Searching

The Basic page allows one to specify a basic search for LDML tags and view the results. LDML 7 preferred tags and their synonyms, and abbreviations will be returned as well as symbols and delimiters.

Figure 2: Basic Search Page

The screenshot shows the 'LDML 7 Reference' interface with the 'Search' tab selected. The 'Basic' sub-tab is active, displaying a search form and a results table.

LDML 7 Tag Search

Tag:

Category:

Type:

Set:

Support:

Sort Results by:

Tags Listing

Name	Type	Set	Support
[Array->Find]	Member	LDML60	Preferred
[Array->Get]	Member	LDML60	Preferred
[Array->Insert]	Member	LDML60	Preferred
[Array->Last]	Member	LDML60	Preferred
[Array->Merge]	Member	LDML60	Preferred
[Array->RemoveAll]	Member	LDML60	Preferred
[Array->Remove]	Member	LDML60	Preferred
[Array->Size]	Member	LDML60	Preferred
[Array->Sort]	Member	LDML60	Preferred
[Array]	Substitution	LDML60	Preferred

search sorted by Name.
Showing 1 to 10 of 10 tags.

Lasso Professional 7 • LDML 7 Reference

© 1996-2003 Blue World Communications, Inc.

Tags can be searched by entering or selecting values from the following fields, and then selecting the **Search** button:

- **Tag** – Specifies the LDML tag by name.
- **Category** – Pull-down menu listing all 30 tag categories.
- **Type** – Pull-down menu listing all possible tag types.
- **Set** – Pull-down menu listing all available tag sets. All preferred Lasso Professional 7 tags belong to the LDML 7.0 set.
- **Support** – Pull-down menu listing the types of tag support in Lasso Professional 7. A **Preferred** tag is part of the core syntax for LDML 7. An **Abbreviation** is an abbreviation of a preferred tag. A **Synonym** is a synonym of a preferred tag. A **Deprecated** tag is supported in LDML 7, but may not be supported in a future version.

Note: Deprecated tags can only be searched using the **Advanced** search page.

- **Sort Results By** – Allows results to be sorted by tag name, type, set, or support.

Selecting the **Find All** button finds all LDML 7 tags in the LDML 7 Reference. Selecting the **Clear** button resets all search fields for a new search.

Search Results

Search results are displayed in the **Tags Listing** panel, which appears to the right. The **Prev** and **Next** buttons are shown if more results are returned than can be shown. Selecting the name of a tag takes one to the ***Detail > Tag*** page for that particular tag.

Advanced Searching

The **Advanced** page provides the same search fields and functionality as the **Basic** page. The results from the **Advanced** page include deprecated and unsupported tags in addition to the preferred tags returned by basic searches.

Figure 3: Advanced Search Page

LDML 7 Tag Search

Tag:

Category:

Type:

Data Source:

Description:

Output Type:

Set:

Support:

Edition:

Version:

Change:

Security Options:

Sort Results by:

Tags Listing

Name	Type	Set	Support
[Array->Find]	Member	LDML60	Preferred
[Array->Get]	Member	LDML60	Preferred
[Array->Insert]	Member	LDML60	Preferred
[Array->Last]	Member	LDML60	Preferred
[Array->Merge]	Member	LDML60	Preferred
[Array->RemoveAll]	Member	LDML60	Preferred
[Array->Remove]	Member	LDML60	Preferred
[Array->Size]	Member	LDML60	Preferred
[Array->Sort]	Member	LDML60	Preferred
[Array]	Substitution	LDML60	Preferred

Search sorted by Name.
Showing 1 to 10 of 10 tags.

Lasso Professional 7 • LDML 7 Reference

© 1996-2003 Blue World Communications, Inc.

Many additional search options are available including:

- **Data Source** – Specifies the data source for which the tag is used.
- **Description** – Allows searching within the tag description.
- **Output Type** – Allows searching for tags that output a value of a particular data type, e.g. Array.
- **Version** – Specifies the version of Lasso from which the tag originated (e.g. 7.0, 6.0, 5.0, 3.6.6.2, etc.).
- **Change** – Specifies whether a tag is new, updated, or unchanged between the last major release and the current release.
- **Security Options** – Specifies whether the tag is controlled by Lasso Security. The options are Classic Lasso for tags that are disabled when Classic Lasso support is disabled, Tag Permissions for tags that can be enabled or disabled by tag permissions, File Permissions for tags that can be enabled or disabled by file permissions, Database Permissions for tags that depend on database or table-level security settings, and LJAPI for tags that are disabled when LJAPI support is disabled.
- **Implementation** – Specifies the implementation of the tag. This can be one of the following:

LDML – Implemented in LDML as part of the in Startup.LassoApp file.

LCAPI – Implemented in C++.

LJAPI – Implemented in Java. Will not work without a JRE installed on the system.

Internal – Implemented in C++ as a core internal language construct. These tags have the lowest-level implementation in LDML.

- **Source Available** – Specifies whether or not the tag source code is available.

Selecting the Find All button finds all LDML 7 tags in the LDML 7 Reference. Selecting the Clear button resets all search fields for a new search.

Search Results

Search results are displayed in the Tags Listing panel, which appears to the right. The Prev and Next buttons are shown if more results are returned than can be shown. Selecting the name of a tag takes one to the *Detail > Tag* page for that particular tag.

Comments Searching

The Comments page allows any of the visitor-entered comments to be searched.

Figure 4: Comments Search Page

LDML 7 Reference blueworld

Search Browse Detail List

Basic Advanced **Comments** Examples Quick Search:

Comment Search

Tag

Author

Subject

Comment

Sort Results by

Comments Listing

Subject	Tag	Date
Uploaded_file_name	[File_Uploads]	02/27/02
Logins a SQL query	[Log] ...	02/27/02
Little_Example	[Decimal->Set...	03/02/02
Value_retains_its_value_for_m...	[Decimal->Set...	03/02/02
Variable_does_not_hold_setfor...	[Decimal->Set...	03/02/02
GroupChar_does_work_with_inte...	[Integer->Set...	03/02/02
client_side_javascript...	[] Square Br...	03/03/02
feature!_client_side_javascr...	[] Square Br...	03/05/02
Refresh_required...	[Database_Cre...	03/09/02
[_datasource_reload]_usage	[Database_Cre...	03/09/02

Search sorted by Name.
Showing 1 to 10 of 33 comments.

Lasso Professional 7 • LDML 7 Reference

© 1996-2003 Blue World Communications, Inc.

The following search options are available.

- **Tag** – Specifies the tag for which the comment was entered.
- **Author** – The name of the author of the comment.
- **Subject** – The subject of each comment can be searched.
- **Comment** – The text of each comment can be searched.

Selecting the Find All button finds all comments which have been entered in the LDML 7 Reference. Selecting the Clear button resets all search fields for a new search.

Search Results

Search results are displayed in the Comments Listing panel, which appears to the right. The Prev and Next buttons are shown if more results are returned than can be shown. Selecting the subject of a comment takes one to the *Detail > Comments* page for that particular comment.

Examples Searching

The Examples page allows any of the tag examples to be searched.

Figure 5: Examples Search Page

LDML 7 Reference **blueworld**

Basic Advanced Comments **Examples** Quick Search:

Example Search

Tag

Title

Description

Example

Results

Sort Results by

Examples Listing

Title	Tag
To create an array of the days of the week;	[Array]
To find an element in an array of pairs;	[Array->Find]
To find and remove an element from an array;	[Array->Remov...
To insert a new element into an array;	[Array->Insert]
To merge two arrays;	[Array->Merge]
To remove an element from an array;	[Array->Remove]
To return an element from an array;	[Array->Get]
To return the last element of an array;	[Array->Last]
To return the size of an array;	[Array->Size]
To sort the elements in an array;	[Array->Sort]

Search sorted by Title.
Showing 1 to 10 of 10 examples.

Lasso Professional 7 • LDML 7 Reference

© 1996-2003 Blue World Communications, Inc.

The following search options are available.

- **Tag** – Specifies the tag for which the example is shown.
- **Title** – The title of each example can be searched.
- **Description** – The description of each example can be searched.
- **Example** – The text of each example's code can be searched.
- **Results** – The text of each example's results can be searched.

Selecting the Find All button finds all examples which have been entered in the LDML 7 Reference. Selecting the Clear button resets all search fields for a new search.

Search Results

Search results are displayed in the Examples Listing panel, which appears to the right. The Prev and Next buttons are shown if more results are returned than can be shown. Selecting the title of an example takes one to the *Detail > Tag* page for that particular example.

Quick Search

A Quick Search field appears in the upper right corner of every page. Entering text in the Quick Search field and pressing Return or Enter on the keyboard performs a basic search on the tag name field and returns results to the Tags Listing panel in the *Search > Basic* page. The last search term entered is displayed in the Quick Search field until a new term is entered or a new search is performed.

Browse

The Browse page allows one to browse the LDML 7 Reference by tag category and tag name for information about LDML tags.

Browsing by Category

The Category page allows one to browse the LDML 7 Reference by tag category and tag name for information about LDML tags.

Figure 6: Category Tags Page

LDML 7 Reference blueworld

Search Browse Detail List

Category Legacy Quick Search:

Tag Categories Listing

Category	Category
Action	Include
Administration	Link
Array	Math
Client	Operator
Conditional	Output
Custom Tag	Response
Data Type	Results
Database	Session
Date	String
Delimiter	Symbol
Encoding	Technical
Encryption	Utility
Error	Variable
File	

Tags Listing for Action

Name	Type	Set	Support
-Add	Command	LDML50, ...	Preferred
-Delete	Command	LDML50, ...	Preferred
-Duplicate	Command	LDML50, ...	Preferred
-FindAll	Command	LDML50, ...	Preferred
-Image	Command	LDML50, ...	Preferred
-Nothing	Command	LDML50, ...	Preferred
-Random	Command	LDML50, ...	Preferred
-Search	Command	LDML50, ...	Preferred
-Show	Command	LDML50, ...	Preferred
-SQL	Command	LDML50, ...	Preferred
-Update	Command	LDML50, ...	Preferred

Showing 1 to 11 of 11 tags.

Lasso Professional 7 • LDML 7 Reference

© 1996-2003 Blue World Communications, Inc.

Viewing Tag Categories

The Tag Categories Listing panel shows a listing of all the 29 tag categories in LDML 7, except legacy tags, which are covered in the next section.

Tags Listing

When a category is selected in the Tag Categories Listing panel, it shows all tags in that category in the Tags Listing panel, which appears to the right. Prev and Next buttons appear for navigation if there are more than ten tags in a selected category. Selecting the name of a tag takes one to the Tag page with the current tag selected, which is described later in this chapter.

Browsing Legacy Tags

The Legacy page allows one to browse all legacy tags in the LDML 7 Reference.

Figure 7: Legacy Tags Page

LDML 7 Reference blueworld

Category **Legacy** Quick Search:

Tag Categories Listing

Category	Category
Administration	Math
Client	Operator
Conditional	Output
Database	Response
Delimiter	Results
Encoding	String
File	Technical
Link	Utility
List	

Tags Listing for Database

Name	Type	Set	Equivalent
-Datasource	Command	LDML25, ...	-Database
-DoScript	Command	LDML25, ...	-FMScript
-DoScript.Post	Command	LDML25, ...	-FMScriptPost
-DoScript.Pre	Command	LDML25, ...	-FMScriptPre
-DoScript.PreSort	Command	LDML25, ...	-FMScriptPreSort
-RecID	Command	LDML25, ...	-KeyValue
-RecordID	Command	LDML3x	-KeyValue
[ChoiceListItem]	Substitution	LDML3x	[Value_ListItem]
[Choice_List]	Substitution	LDML3x	
[Choice_List] ...	Container	LDML3x	
[Datasource_Name]	Substitution	LDML3x	[Database_Name]
[DB_LayoutNameItem]	Substitution	LDML3x	[Database_TableNameItem]
[DB_LayoutNames] ...	Container	LDML3x	[Database_TableNames] ...
[DB_NameItem]	Substitution	LDML3x	[Database_NameItem]
[DB_Names] ...	Container	LDML3x	[Database_Names] ...

Showing 1 to 15 of 21 tags. Next >

Lasso Professional 7 • LDML 7 Reference

© 1996-2003 Blue World Communications. Inc.

Legacy tags include all deprecated tags from LDML 3 and earlier. As support for select legacy tags may be dropped in future releases of Lasso Professional, using these tags to build Lasso solutions is not recommended.

One is able to browse all legacy tags in the Legacy page in the same manner as in the Category page, covered in the previous section.

Detail


The Detail section shows information and comments about any selected tag.

Tag Detail

The Tag page shows all information about a selected tag. One is taken here after selecting a tag from the Search, Browse, or List sections. All information is shown in the left panel, and includes the following:

- **Description** – Defines what a tag does, and how and where it is used.
- **Syntax** – Shows the syntax for the tag.
- **Parameters** – Lists all parameters or modifiers that can be used with the tag. Required Parameters must be present in the tag syntax for the tag to work properly, while Optional Parameters do not.
- **Examples** – Provides examples of how the tag can be used to perform a specific function within a Lasso solution.
- **Change Notes** – Provides information about how a tag has changed from different versions of Lasso, and if applicable, what tag it replaces.

Figure 8: Tag Detail Page



Search

Browse

Detail

List

blueworld

Tag

Comments

Code

Quick Search:

[Array->Find]

Next >

Description

[Array->Find] returns an array of elements that match the parameter. Accepts a single parameter of any data type.

If the array contains any pair values, only the first part of the pair is compared with the parameter of the [Array->Find] tag.

If no elements in the array match the parameter to the [Array->Find] tag then an empty array is returned.

Syntax

[Array->(Find: 'Find Value')]

Parameters

Required Parameters

Array

The array which should be searched.

Find Value

The value which should be searched for in the array.

Examples

To find an element in an array of pairs:

Use the [Array->Find] tag with the value of the first element of the pairs that should be returned from the array. The following example shows an array of pairs returned from the tag, each of which has a first element of 'John Doe'.

[Var: 'People_Array'=(Array: 'John Doe'='Person One', 'Jane Doe'='Person Two', 'Joe Surname'='Person Three', 'John Doe'='Person Four')]

[Output: \$People_Array->(Find: 'John Doe')]

→ (Array: (Pair: (John Doe)=(Person One), (John Doe)=(Person Four)))

Tag Link

[\[Array->Find\]](#)

Category

[Array](#)

Type

Member

Set

LDML60

Support

Preferred

Edition

Standard, Developer

Version

6.0

Change

New

Data Source

Any

Output Type

Any

Security

None

Page Number

294

Comments

0

Related Tags

[\[Array->Get\]](#)

[\[Array\]](#)

[\[Map->Find\]](#)

Lasso Professional 7 • LDML 7 Reference

© 1996-2003 Blue World Communications. Inc.

The top panel shows the current tag selected. If a search was performed, one can navigate through the found set by selecting the *Prev* and *Next* buttons. If no search was performed, the *Prev* and *Next* buttons will navigate through the tags in each category alphabetically.

The right panel lists the following tag information:

- **Category** – Specifies the tag category (e.g. Array, Encoding, etc.). Selecting the tag category displays the *Browse > Category* page.
- **Type** – Specifies the tag type (e.g. Command, Container, etc.).
- **Set** – Specifies the versions of LDML in which the tag is supported. All native Lasso Professional 7 tags belong to the LDML 7.0 set.
- **Support** – Specifies the tag support in Lasso Professional 7. A Preferred tag is part of the core syntax for LDML 7. An Abbreviation is an abbreviation of a preferred tag. A Synonym is a synonym of a preferred tag. A Deprecated tag is supported in LDML 7, but support may be dropped in a future version of Lasso. Deprecated tags are not recommended for use in new projects. Any returns all support types.
- **Version** – Specifies the version of Lasso from which the tag originated (e.g. 7.0, 6.0, 5.0, 3.6.6.2, etc.).
- **Change** – Specifies whether a tag is new, updated, or unchanged between the last major release and the current release.
- **Data Source** – Specifies the data source with which the tag can be used.
- **Output Type** – Specifies what data type the tag will output. Many tags output multiple data types in which case each data type or Any is shown.
- **Security** – Specifies whether access to the tag can be controlled through Lasso Administration. Options include Classic for tags that are disabled with Classic Lasso, Tag for tags that are controlled by tag permissions, File for tags that are controlled by file permissions, Database for tags that are controlled by database permissions, and LJAPI for tags that are disabled if LJAPI support is disabled.
- **Page Number** – Specifies what page number in the Lasso 7 Language Guide contains the primary reference for the tag. Some tags are also documented in the Extending Lasso Guide. These tags are marked ELG.
- **Comments** – Indicates the number of comments that have been entered for the tag. Selecting the link takes the visitor to the *Comments* page.

The lower right panel contains links to other tags in the database. The following types of tags are listed.

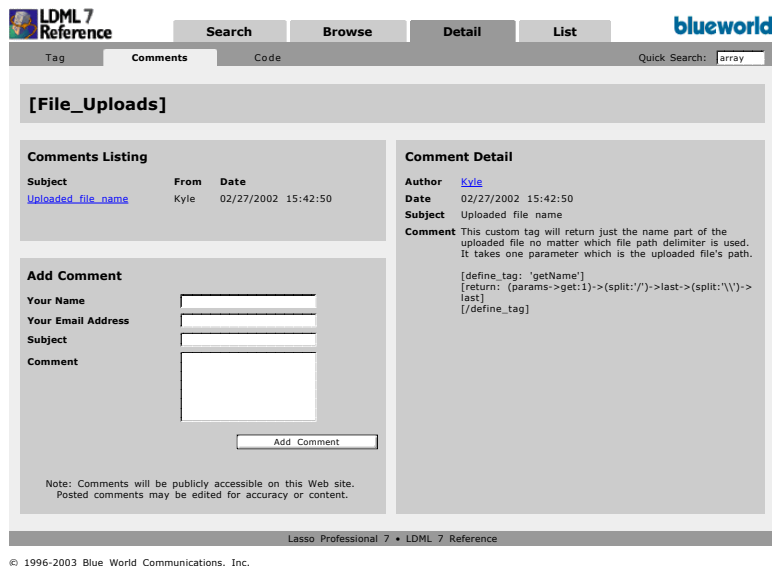
- **Synonyms** – Lists any tags that are synonyms of the current tag. Synonyms accept the same parameters and can be used interchangeably.
- **Abbreviations** – Lists any abbreviations for the current tag.

- **Related Tags** – Lists any related tags, which are tags that have similar functions or are used in a similar manner.
- **Required Tags** – Lists all tags and technologies (e.g. Java) that are required for the selected tag to work.
- **LDML 3 Equivalent** – For tags which have been updated since LDML 3, an LDML 3 tag is listed that provides similar functionality to the current tag.
- **LDML 7 Equivalent** – For tags which are not preferred LDML 7 syntax, an equivalent LDML 7 tag is listed that provides similar functionality to the current tag.

Tag Comments

The Comments page allows users to add their own notes and comments about a tag to the LDML Reference. The top panel shows the current tag selected. One can navigate through the tags alphabetically by selecting the Prev and Next buttons.

Figure 9: Tag Comments Page



Comments Listing

The Comments Listing panel shows a list of all comments about a selected tag. Prev and Next buttons appear at the bottom of this panel for navigation.

if there are more than five comments. Selecting the subject name under Subject shows the comment in the Comment Detail panel. Selecting the name of the author under From will display the author's email address. The date and time of when the comment was added is shown under Date.

Adding Comments

The Add Comment panel allow users of the LDML Reference LassoApp to add comments about the selected tag.

To add a comment:

- 1 Enter your name in the Your Name field.
- 2 Enter your email address in the Your Email field.
- 3 Enter the subject of the comment in the Subject field.
- 4 Enter your comment in the Comment field.
- 5 Select Add Comment.

Comment Detail

The Comment Detail panel displays the comment author, the date of the comment, the subject, and the comment.

The LDML 7 Reference stores all user comments locally, and only comments from users of the LDML Reference LassoApp on that machine are shown. A version of the LDML 7 Reference can also be accessed on the Blue World Web site at:

<http://ldml.blueworld.com/>

This version contains the same information as the locally-stored LDML 7 Reference, however it also contains comments from other users and developers. This is useful for finding further examples and information about particular tags.

Legacy Tags

The Legacy page provides a listing of all legacy tags, which are deprecated tags from LDML 6, LDML 5, and LDML 3.

Figure 11: Legacy Tags Page

The screenshot shows the 'LDML 7 Legacy Tags' page. At the top, there are tabs for 'Preferred', 'Legacy' (selected), and 'All'. A search bar is on the right. The page is divided into two main panels. The left panel lists legacy tags under four categories: Administration, Client, Conditional, and Database. The right panel, titled 'Database (Continued)', lists more database-related legacy tags. Each tag is shown with its name, type, set, and equivalent tags.

Category	Name	Type	Set	Equivalent
Administration	[Lasso_Datasources4D]	Substitution	LDML3x	
	[Lasso_DatasourcesODBC]	Substitution	LDML3x	
	[Lasso_DataType]	Substitution	LDML3x	[Null->Type]
Client	[Client_Addr]	Substitution	LDML1x, ...	[Client_Address]
Conditional	[Else.If]	Substitution	LDML25, ...	[Else]
	[Lasso_Abort]	Process	LDML3x	[Abort]
	[LoopAbort]	Process	LDML3x	[Loop_Abort]
	[LoopCount]	Substitution	LDML3x	[Loop_Count]
Database	.Database	Command	LDML25, ...	-Database
	.DoScript	Command	LDML25, ...	-FMScript
Database (Continued)	.DoScript.Post	Command	LDML25, ...	-FMScriptPost
	.DoScript.Pre	Command	LDML25, ...	-FMScriptPre
	.DoScript.PreSort	Command	LDML25, ...	-FMScriptPreSort
	.RecID	Command	LDML25, ...	-KeyValue
	.RecordID	Command	LDML3x	-KeyValue
	[ChoiceListItem]	Substitution	LDML3x	[Value_ListItem]
	[Choice_ListItem]	Substitution	LDML3x	
	[Choice_List]	Container	LDML3x	
	[Database_Name]	Substitution	LDML3x	[Database_Name]
	[DB_LayoutNameItem]	Substitution	LDML3x	[Database_TableNameItem]

Navigation buttons: Prev, Next >

Footer: Lasso Professional 7 • LDML 7 Reference

All tags are listed alphabetically beneath their category name (e.g. Array, Database, etc.) and the list spans both panels. The listing can be navigated by selecting the Prev and Next buttons at the top of the page. Selecting a tag name takes one to the Tag page, covered earlier in this chapter.

All Tags

The All page provides a listing of all LDML tags available in LDML 7 including preferred tags and legacy tags. All tags are listed alphabetically, and span both panels. The listing can be navigated by selecting the Prev and Next buttons at the top of the page. Selecting a tag name takes one to the Tag page, covered earlier in this chapter.



Section II

Database Interaction

This section includes an introduction to interacting with databases in Lasso Professional 7 and more specific discussions of particular database actions and tags and techniques particular to Lasso MySQL and FileMaker Pro databases.

- *Chapter 6: Database Interaction Fundamentals* introduces the concepts required to work with databases in Lasso Professional 7.
- *Chapter 7: Searching and Displaying Data* discusses how to create search queries and display the results of those queries.
- *Chapter 8: Adding and Updating Records* discusses how to create queries to add, update, and delete database records.
- *Chapter 9: MySQL Data Sources* documents tags specific to the Lasso MySQL data source connector and MySQL data source connector including tags to create database schema programmatically.
- *Chapter 10: FileMaker Pro Data Sources* documents tags specific to the FileMaker Pro data source connector including tags to execute FileMaker Pro scripts, return images from a FileMaker Pro database, and display information in repeating fields and portals.
- *Chapter 11: JDBC Pro Data Sources* documents tags specific to the JDBC data source connector.

6

Chapter 6

Database Interaction Fundamentals

One of the primary purposes of LDML is to perform database actions which are a combination of pre-defined and visitor-defined parameters and to format the results of those actions. This chapter introduces the fundamentals of specifying database actions in LDML.

- *Inline Database Actions* includes full details for how to use the `[Inline]` tag to specify database actions.
- *Action Parameters* describes how to get information about an action.
- *Results* includes information about how to return details of an LDML database action.
- *Showing Database Schema* describes the tags that can be used to examine the schema of a database.
- *SQL Statements* describes the `-SQL` command tag and how to issue raw SQL statements to SQL-compliant data sources.
- *SQL Transactions* describes how to perform reversible SQL transactions using Lasso.

Inline Database Actions

The `[Inline] ... [/Inline]` container tags are used to specify a database action and to present the results of that action within a Lasso format file. The database action is specified using parameters as keyword/value parameters within the opening `[Inline]` tag. Additional name/value parameters specify the user-defined parameters of the database action. A single action can be specified

in an [Inline]. Additional actions can be performed in subsequent or nested [Inline] ... [/Inline] tags.

Table 1: Inline Tag

Tag	Description
[Inline] ... [/Inline]	Performs the database action specified in the opening tag. The results of the database action are available inside the container tag.

The results of the database action can be displayed within the contents of the [Inline] ... [/Inline] container tags using the [Records] ... [/Records] container tags and the [Field] substitution tag. Alternately, the [Inline] can be named and the results can be displayed later.

The entire database action can be specified directly in the opening [Inline] tag or visitor-defined aspects of the action can be retrieved from an HTML form submission. [Link_...] tags can be used to navigate a found set in concert with the use of [Inline] ... [/Inline] tags. Nested [Inline] ... [/Inline] tags can be used to create complex database actions.

Database Actions

A database action is performed to retrieve data from a database or to manipulate data which is stored in a database. Database actions can be used in Lasso to query records in a database that match specific criteria, to return a particular record from a database, to add a record to a database, to delete a record from a database, to fetch information about a database, or to navigate through the found set from a database search. In addition, database actions can be used to execute SQL statements in compliant databases.

The database actions in Lasso are defined according to what action parameter is used to trigger the action. The following table lists the parameters which perform database actions that are available in LDML.

Table 2: Inline Database Action Parameters

Tag	Description
-Search	Finds records in a database that match specific criteria, returns detail for a particular record in a database, or navigates through a found set of records.
-FindAll	Returns all records in a specific database table.
-Random	Returns a single, random record from a database table.
-Add	Adds a record to a database table.
-Update	Updates a specific record from a database table.

-Duplicate	Duplicates a specific record in a database table. Only works with FileMaker Pro databases.
-Delete	Removes a specified record from a database table.
-Show	Returns information about the tables and fields within a database.
-SQL	Executes a SQL statement in a compatible data source. Only works with Lasso MySQL and other SQL databases.
-Nothing	The default action which performs no database interaction, but simply passes the parameters of the action.

Note: *Table 2: Database Action Parameters* lists all of the database actions that Lasso supports. Individual data source connectors may only support a subset of these parameters. The Lasso Connector for Lasso MySQL and the Lasso Connector for MySQL do not support the **-Duplicate** action. The Lasso Connector for FileMaker Pro does not support the **-SQL** action. See the documentation for third party data source connectors for information about what parameters they support.

Each database action parameter requires additional parameters in order to execute the proper database action. These parameters are specified using additional parameters and name/value pairs. For example, a **-Database** parameter specifies the database in which the action should take place and a **-Table** parameter specifies the specific table from that database in which the action should take place. Name/value pairs specify the query for a **-Search** action, the initial values for the new record created by an **-Add** action, or the updated values for an **-Update** action.

Full documentation of which [Inline] parameters are required for each action are detailed in the section specific to that action in this chapter, *Chapter 7: Searching and Displaying Data*, or *Chapter 8: Adding and Updating Records*.

Example of specifying a **-FindAll** action within an [Inline]:

The following example shows an [Inline] ... [/Inline] tag that has a **-FindAll** database action specified in the opening tag. The [Inline] tag includes a **-FindAll** parameter to specify the action, **-Database** and **-Table** parameters to specify the database and table from which records should be returned, and a **-KeyField** parameter which specifies the key field for the table. The entire database action is hard-coded within the [Inline] tag.

The tag [Found_Count] returns how many records are in the database. The [Records] ... [/Records] container tags repeat their contents for each record in

the found set. The [Field] tags are repeated for each found record creating a listing of the names of all the people stored in the Contacts database.

```
[Inline: -FindAll,
-Database='Contacts',
-Table='People',
-KeyField='ID']
There are [Found_Count] record(s) in the People table.
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

→ There are 2 record(s) in the People table.
John Doe
Jane Doe

Example of specifying a -Search action within an [Inline]:

The following example shows an [Inline] ... [/Inline] tag that has a -Search database action specified in the opening tag. The [Inline] tag includes a -Search parameter to specify the action, -Database and -Table parameters to specify the database and table records from which records should be returned, and a -KeyField parameter which specifies the key field for the table. The subsequent name/value parameters, 'First_Name'='John' and 'Last_Name'='Doe', specify the query which will be performed in the database. Only records for John Doe will be returned. The entire database action is hard-coded within the [Inline] tag.

The tag [Found_Count] returns how many records for John Doe are in the database. The [Records] ... [/Records] container tags repeat their contents for each record in the found set. The [Field] tags are repeated for each found record creating a listing of all the records for John Doe stored in the Contacts database.

```
[Inline: -Search,
-Database='Contacts',
-Table='People',
-KeyField='ID',
'First_Name'='John',
'Last_Name'='Doe']
There were [Found_Count] record(s) found in the People table.
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

→ There were 1 record(s) found in the People table.
John Doe

Using HTML Forms

The previous two examples show how to specify a hard-coded database action completely within an opening `[Inline]` tag. This is an excellent way to embed a database action that will be the same every time a page is loaded, but does not provide any room for visitor interaction.

A more powerful technique is to use values from an HTML form or URL to allow a site visitor to modify the database action which is performed within the `[Inline]` tag. The following two examples demonstrate two different techniques for doing this using the singular `[Action_Param]` tag and the array-based `[Action_Params]` tag.

Example of using HTML form values within an `[Inline]` with `[Action_Param]`:

An inline-based database action can make use of visitor specified parameters by reading values from an HTML form which the visitor customizes and then submits to trigger the page containing the `[Inline]` ... `[/Inline]` tags.

The following HTML form provides two inputs into which the visitor can type information. An input is provided for `First_Name` and one for `Last_Name`. These correspond to the names of fields in the `Contacts` database. The action of the form is set to `response.lasso` which will contain the `[Inline]` ... `[/Inline]` tags that perform the actual database action. The action tag specified in the form is `-Nothing` which instructs Lasso to perform no database action when the form is submitted.

```
<form action="/response.lasso" method="POST">
  <br>First Name: <input type="text" name="First_Name" value="">
  <br>Last Name: <input type="text" name="Last_Name" value="">
  <br><input type="submit" value="Search">
</form>
```

The `[Inline]` tag on `response.lasso` contains the name/value parameter `'First_Name'=(Action_Param: 'First_Name')`. The `[Action_Param]` tag instructs Lasso to fetch the input named `First_Name` from the action which resulted in the current page being served, namely the form shown above. The `[Inline]` contains a similar name/value parameter for `Last_Name`.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'=(Action_Param: 'First_Name'),
  'Last_Name'=(Action_Param: 'Last_Name')]
There were [Found_Count] record(s) found in the People table.
```

```
[Records]
<br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

If the visitor entered Jane for the first name and Doe for the last name then the following results would be returned.

→ There were 1 record(s) found in the People table.
Jane Doe

As many parameters as are needed can be named in the HTML form and then retrieved in the response page and incorporated into the [Inline] tag.

Note: The [Action_Param] tag is equivalent to the [Form_Param] tag used in prior versions of Lasso.

Example of using an array of HTML form values within an [Inline] with [Action_Params]:

Rather than specifying each [Action_Param] individually, an entire set of HTML form parameters can be entered into an [Inline] tag using the array-based [Action_Params] tag. Inserting the [Action_Params] tag into an [Inline] functions as if all the parameters and name/value pairs in the HTML form were placed into the [Inline] at the location of the [Action_Params] parameter.

The following HTML form provides two inputs into which the visitor can type information. An input is provided for First_Name and one for Last_Name. These correspond to the names of fields in the Contacts database. The action of the form is set to response.lasso which will contain the [Inline] ... [/Inline] tags that perform the actual database action. The database action is -Nothing which instructs Lasso to perform no database action when the HTML form is submitted.

```
<form action="/response.lasso" method="POST">
  <br>First Name: <input type="text" name="First_Name" value="">
  <br>Last Name: <input type="text" name="Last_Name" value="">
  <br><input type="submit" value="Search">
</form>
```

The [Inline] tag on response.lasso contains the array parameter [Action_Params]. This instructs Lasso to take all the parameters from the HTML form or URL which results in the current page being loaded and insert them in the [Inline] as if they had been typed at the location of [Action_Params]. This will result in the name/value pairs for First_Name, Last_Name, and the -Nothing action to be inserted into the [Inline]. The latest action specified has precedence so the -Search tag specified in the actual [Inline] tag overrides the -Nothing which is passed from the HTML form.

```
[Inline: (Action_Params),
-Search,
-Database='Contacts',
-Table='People',
-KeyField='ID']
There were [Found_Count] record(s) found in the People table.
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

If the visitor entered Jane for the first name and Doe for the last name then the following results would be returned.

→ There were 1 record(s) found in the People table.
Jane Doe

As many parameters as are needed can be named in the HTML form. They will all be incorporated into the [Inline] tag at the location of the [Action_Params] tag. Any parameters in the [Inline] after the [Action_Params] tag will override conflicting settings from the HTML form.

Note: [Action_Params] is a replacement for the -ReUseFormParams keyword in prior versions of Lasso. See *Chapter 31: Upgrading Your Solutions* for more information.

HTML Form Response Pages

Every HTML form or URL needs to have a response page specified so Lasso knows what format file to process and return as the result of the action. The referenced format file could contain simple HTML or complex LDML calculations, but some format file must be specified.

To specify a format file within an HTML form or URL:

- The HTML form action can be set to the location of a format file. For example, the following HTML <form> tag references the file /response.lasso in the root of the Web serving folder.

```
<form action="/response.lasso" method="POST"> ... </form>
```

- The URL can reference the location of a format file before the question mark ? delimiter. For example, the following anchor tag references the file response.lasso in the same folder as the page in which this anchor is contained.

```
<a href="response.lasso?Name=Value"> Link </a>
```

- The HTML form can reference /Action.Lasso and then specify the path to the format file in a -Response tag. For example, the following HTML

`<form>` tag references the file `response.lasso` in the root of the Web serving folder. The path is relative to the root because the placeholder `/Action.Lasso` is specified with a leading forward slash `/`.

```
<form action="/Action.Lasso" method="POST">
  <input type="hidden" name="-Response" value="response.lasso">
</form>
```

- The URL can reference `Action.Lasso` and then specify the path to the format file in a `-Response` tag. For example, the following anchor tag references the file `response.lasso` in the same folder as the page in which the link is specified. The path is relative to the local folder because the placeholder `Action.Lasso` is specified without a leading forward slash `/`.

```
<a href="Action.Lasso?-Response=response.lasso"> Link </a>
```

The `-Response` tag can be used on its own or action specific response tags can be used so a form is sent to different response pages if different actions are performed using the form. Response tags can also be used to send the visitor to different pages if different errors happen when the database action is attempted by Lasso. The following table details the available response tags.

Table 3: Response Parameters

Tag	Description
<code>-Response</code>	Default response tag. The value for this response tag is used if no others are specified.
<code>-ResponseAnyError</code>	Default error response tag. The value for this response tag is used if any error occurs and no more specific error response tag is set.
<code>-ResponseReqFieldMissingError</code>	Error to use if a <code>-Required</code> field is not given a value by the visitor.
<code>-ResponseSecurityError</code>	Error to use if a security violation occurs because the current visitor does not have permission to perform the database action.

See *Chapter 21: Error Control* for more information about using the error response pages.

Setting HTML Form Values

If the format file containing an HTML form is the response to an HTML form or URL, then the values of the HTML form inputs can be set to values retrieved from the previous format file using `[Action_Param]`.

For example, if a form is on `default.lasso` and the action of the form is `default.lasso` then the same page will be reloaded with new form values each

time the form is submitted. The following HTML form uses [Action_Param] tags to automatically restore the values the user specified in the form previously, each time the page is reloaded.

```
<form action="default.lasso" method="POST">
  <br>First Name:
    <input type="hidden" name="First_Name" value="[Action_Param: 'First_Name']">
  <br>First Name:
    <input type="hidden" name="Last_Name" value="[Action_Param: 'Last_Name']">
  <br><input type="submit" value="Submit">
</form>
```

Tokens

Tokens can be used with HTML forms and URLs to order to pass data along with the action. Tokens are useful because they do not affect the operation of a database action, but allow data to be passed along with the action. For example, meta-data could be associated with a visitor to a Web site without using sessions or cookies.

- Tokens can be set in a form using the -Token.TokenName=TokenValue parameter. Multiple named tokens can be set in a single form.

```
<form action="response.lasso" method="POST">
  <input type="hidden" name="-Token.TokenName" value="TokenValue">
</form>
```

- Tokens can be set in a URL using the -Token.TokenName=TokenValue parameter. Multiple named tokens can be set in a single URL.

```
<a href="response.lasso?-Token.TokenName=TokenValue"> Link </a>
```

- Tokens set in an HTML form or URL are available in the response page of the database action. Tokens are not available inside [Inline] ... [/Inline] tags on the responses page unless they are explicitly set within the [Inline] tag itself.
- Tokens can be set in an [Inline] using the -Token.TokenName=TokenValue parameter. Multiple named tokens can be set in a single [Inline].
- Tokens set in an [Inline] are only available immediately inside the [Inline]. They are not available to nested [Inlines] unless they are set specifically within each [Inline].
- By default, tokens are included in the [Link_...] tags and in [Action_Params]. Unless specifically set otherwise, tokens will be redefined on pages which are returned using the [Link_...] tags.

Nesting Inline Database Actions

Database actions can be combined to perform compound database actions. All the records in a database that meet certain criteria could be updated or deleted. Or, all the records from one database could be added to a different database. Or, the results of searches from several databases could be merged and used to search another database.

Database actions are combined by nesting `[Inline] ... [/Inline]` tags. For example, if `[Inline] ... [/Inline]` tags are placed inside the `[Records] ... [/Records]` container tag within another set of `[Inline] ... [/Inline]` tags then the inner `[Inline]` will execute once for each record found in the outer `[Inline]`.

All database results tags function for only the innermost set of `[Inline] ... [/Inline]` tags. Variables can pass through into nested `[Inline] ... [/Inline]` tags, but tokens cannot, these need to be reset in each `[Inline]` tag in the hierarchy.

SQL Note: Nested inlines can also be used to perform reversible SQL transactions in transaction-compliant SQL data sources. See the *SQL Transactions* section at the end of this chapter for more information.

Example of nesting `[Inline] ... [/Inline]` tags:

This example will use nested `[Inline] ... [/Inline]` tags to change the last name of all people whose last name is currently Doe in a database to Person. The outer `[Inline] ... [/Inline]` tags perform a hard-coded search for all records with `Last_Name` equal to Doe. The inner `[Inline] ... [/Inline]` tags update each record so `Last_Name` is now equal to Person. The output confirms that the conversion went as expected by outputting the new values.

```
[Inline: -Search,
-Database='Contacts',
-Table='People',
-KeyField='ID',
'Last_Name'='Doe',
-MaxRecords='All']
[Records]
[Inline: -Update,
-Database='Contacts',
-Table='People',
-KeyField='ID',
-KeyValue=(KeyField_Value),
'Last_Name'='Person']
<br>Name is now [Field: 'First_Name'] [Field: 'Last_Name']
[/Inline]
[/Records]
[/Inline]
```

→ Name is now Jane Person
 Name is now John Person

Array Inline Parameters

Most LDML parameter can be used within an [Inline] tag to specify an action. In addition, parameters and name/value parameters can be stored in an array and then passed into an [Inline] as a block. Any single value in an [Inline] which is an array data type will be interpreted as a series of parameters inserted at that location in the array. This technique is useful for programmatically assembling database actions.

Many parameters can only take a single value within an [Inline] tag. For example, only a single action can be specified and only a single database can be specified. The last action parameter defines the value that will be used for the action. The last, for example, -Database parameter defines the value that will be used for the database of the action. If an array parameter comes first in an [Inline] then all subsequent parameters will override any conflicting values within the array parameter.

Example of using an array to pass values into an [Inline]:

The following LassoScript performs a -FindAll database action with the parameters first specified in an array and stored in the variable Params, then passed into the opening [Inline] tag all at once. The value for -MaxRecords in the [Inline] tag overrides the value specified within the array parameter since it is specified later. Only the number of records found in the database are returned using the [Output] tag.

```
<?LassoScript
  Variable: 'Params'=(Array:
    -FindAll=",
    -Database='Contacts',
    -Table='People',
    -MaxRecords=50
  );
  Inline: (Var: 'Params'), -MaxRecords=100;
  Output: 'There are ' + (Found_Count) + 'record(s) in the People table.';
/Inline;
?>
```

→ There are 2 record(s) in the People table.

Action Parameters

LDML has a set of substitution tags which allow for information about the current action to be returned. The parameters of the action itself can be returned or information about the action's results can be returned.

The following table details the substitution tags which allow information about the current action to be returned. If these tags are used within an [Inline] ... [/Inline] container tag they return information about the action specified in the opening [Inline] tag. Otherwise, these tags return information about the action which resulted in the current format file being served.

Even format files called with a simple URL such as <http://www.example.com/response.lasso> have an implicit -Nothing action. Many of these substitution tags return default values even for the -Nothing action.

Table 4: Action Parameter Tags

Tag	Description
[Action_Param]	Returns the value for a specified name/value parameter. Equivalent to [Form_Param].
[Action_Params]	Returns an array containing all of the parameters and name/value parameters that define the current action.
[Database_Name]	Returns the name of the current database.
[KeyField_Name]	Returns the name of the current key field.
[KeyField_Value]	Returns the name of the current key value if defined. Equivalent to [RecordID_Value].
[Lasso_CurrentAction]	Returns the name of the current Lasso action.
[MaxRecords_Value]	Returns the number of records from the found set that are currently being displayed.
[Operator_LogicalValue]	Returns the value for the logical operator.
[Response_FilePath]	Returns the path to the current format file.
[SkipRecords_Value]	Returns the current offset into a found set.
[Table_Name]	Returns the name of the current table. Equivalent to [Layout_Name].
[Token_Value]	Returns the value for a specified token.
[Search_Arguments]	Container tag repeats once for each name/value parameter of the current action.
[Search_FieldItem]	Returns the name portion of a name/value parameter of the current action.
[Search_OperatorItem]	Returns the operator associated with a name/value parameter of the current action.
[Search_ValueItem]	Returns the value portion of a name/value parameter of the current action.

[Sort_Arguments]	Container tag repeats once for each sort parameter.
[Sort_FieldItem]	Returns the field which will be sorted.
[Sort_OrderItem]	Returns the order by which the field will be sorted.

The individual substitution tags can be used to return feedback to site visitors about what database action is being performed or to return debugging information. For example, the following code inserted at the top of a response page to an HTML form or URL or in the body of an [Inline] ... [/Inline] tag will return details about the database action that was performed.

```
Action: [Lasso_CurrentAction]
Database: [Database_Name]
Table: [Table_Name]
Key Field: [KeyField_Name]
KeyValue: [KeyField_Value]
MaxRecords: [MaxRecords_Value]
SkipRecords: [SkipRecords_Value]
Logical Operator: [Operator_LogicalValue]
```

```
→ Action: Find All
Database: Contacts
Table: People
Key Field: ID
KeyValue: 100001
MaxRecords: 50
SkipRecords: 0
Logical Operator: AND
```

The [Action_Params] tag can be used to return information about the entire Lasso action in a single array. Rather than assembling information using the individual substitution tags it is often easier to extract information from the [Action_Params] array. The schema of the array returned by [Action_Params] is detailed in *Table 5: [Action_Params] Array Schema*.

The schema shows the names of the values which are returned in the array. Even if -Layout is used to specify the layout for a database action, the value of that tag is returned after -Table in the [Action_Params] array.

To output the parameters of the current database action:

The value of the [Action_Params] tag in the following example is formatted to show the elements of the returned array clearly. The [Action_Params] array contain values for -MaxRecords, -SkipRecords, and -OperatorLogical even though these aren't specified in the [Inline] tag.

```

[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
[Action_Params]
[/Inline]

```

```

→ (Array:
  (Pair: (-Search) = ()),
  (Pair: (-Database) = (Contacts)),
  (Pair: (-Table) = (People)),
  (Pair: (-KeyField) = (ID)),
  (Pair: (-MaxRecords) = (50)),
  (Pair: (-SkipRecords) = (0)),
  (Pair: (-OperatorLogical) = (AND))
)

```

Table 5: [Action_Params] Array Schema

Name	Description
Action	The action parameter is always returned first. The name of the first item is set to the action parameter and the value is left empty.
-Database	If defined, the name of the current database.
-Table	If defined, the name of the current table.
-KeyField	If defined, the name of the field which holds the primary key for the specified table.
-KeyValue	If defined, the particular value for the primary key.
-MaxRecords	Always included. Defaults to 50.
-SkipRecords	Always included. Defaults to 0.
-OperatorLogical	Always included. Defaults to AND.
-ReturnField	If defined, can have multiple values.
-SortOrder, -SortField	If defined, can have multiple values. -SortOrder is always defined for each -SortField. Defaults to ascending.
-Token	If defined, can have multiple values each specified as -Token.TokenName with the appropriate value.
Name/Value Parameter	If defined, each name/value parameter is included.
-Required	If defined, can have multiple values. Included in order within name/value parameters.
-Operator	If defined, can have multiple values. Included in order within name/value parameters.
-OperatorBegin	If defined, can have multiple values. Included in order within name/value parameters.
-OperatorEnd	If defined, can have multiple values. Included in order within name/value parameters.

The [Action_Params] array contains all the parameters and name/value parameters required to define a database action. It does not include any -Response... parameters, the -Username and -Password parameters, -FMScript... parameters, -InlineName keyword or any legacy or unrecognized parameters.

To output the name/value parameters of the current database action:

Loop through the [Action_Params] tag and display only name/value pairs for which the name does not start with a hyphen, i.e. any name/value pairs which do not start with a keyword. The following example shows a search of the People table of the Contacts database for a person named John Doe.

```

[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'='John',
  'Last_Name'='Doe']
[Loop: (Action_Params)->Size]
  [If: !(Action_Params)->(Get: Loop_Count)->(First)->(BeginsWith: '-')]
    <br>[Output: (Action_Params)->(Get: Loop_Count)]
  [/If]
[/Loop]
[/Inline]
→ <br>(Pair: (First_Name) = (John))
  <br>(Pair: (Last_Name) = (Doe))

```

To display action parameters to a site visitor:

The [Search_Arguments] ... [/Search_Arguments] container tag can be used in conjunction with the [Search_FieldItem], [Search_ValueItem] and [Search_OperatorItem] substitution tags to return a list of all name/value parameters and associated operators specified in a database action.

```

[Search_Arguments]
  <br>[Search_OperatorItem] [Search_FieldItem] = [Search_ValueItem]
[/Search_Arguments]

```

The [Sort_Arguments] ... [/Sort_Arguments] container tag can be used in conjunction with the [Sort_FieldItem] and [Sort_OrderItem] substitution tags to return a list of all name/value parameters and associated operators specified in a database action.

```

[Sort_Arguments]
  <br>[Sort_OperatorItem] [Sort_FieldItem] = [Sort_ValueItem]
[/Sort_Arguments]

```

Results

The following table details the substitution tags which allow information about the results of the current action to be returned. These tags provide information about the current found set rather than providing data from the database or providing information about what database action was performed.

Table 6: Results Tags

Tag	Description
[Field]	Returns the value for a specified field from the result set.
[Found_Count]	Returns the number of records found by Lasso.
[Records] ... [/Records]	Loops once for each record in the found set. [Field] tags within the [Records] ... [/Records] tags will return the value for the specified field in each record in turn.
[Records_Array]	Returns the complete found set in an array of arrays. The outer array contains one item for every record in the found set. The item for each record is an array containing one item for each field in the result set.
[Shown_Count]	Returns the number of records shown in the current found set. Less than or equal to [MaxRecords_Value].
[Shown_First]	Returns the number of the first record shown from the found set. Usually [SkipRecords_Value] plus one.
[Shown_Last]	Returns the number of the last record shown from the found set.
[Total_Records]	Returns the total number of records in the current table. Works with FileMaker Pro databases only.

The found set tags can be used to display information about the current found set. For example, the following code generates a status message that can be displayed under a database listing.

```
Found [Found_Count] records of [Total_Records] Total.
<br>Displaying [Shown_Count] records from [Shown_First] to [Shown_Last].
```

→ Found 100 records of 1500 Total.
Displaying 10 records from 61 to 70.

These tags can also be used to create links that allow a visitor to navigate through a found set. See *Chapter 7: Searching and Displaying Data* for more information.

Showing Database Schema

The schema of a database can be inspected using the [Database_...] tags or the -Show parameter which allows information about a database to be returned using the [Field_Name] tag. Value lists within FileMaker Pro databases can also be accessed using the -Show parameter. This is documented fully in *Chapter 10: FileMaker Pro Data Sources*.

Table 7: -Show Parameter

Tag	Description
-Show	Allows information about a particular database and table to be retrieved.

The -Show parameter functions like the -Search parameter except that no name/value parameters, sort tags, results tags, or operator tags are required. -Show actions can be specified in [Inline] ... [/Inline] tags, HTML forms, or URLs.

Table 8: -Show Action Requirements

Tag	Description
-Show	The action which is to be performed. Required.
-Database	The database which should be searched. Required.
-Table	The table from the specified database which should be searched. Required.
-KeyField	The name of the field which holds the primary key for the specified table. Recommended.

The tags detailed in *Table 9: Schema Tags* allow the schema of a database to be inspected. The [Field_Name] tag must be used in concert with a -Show action or any database action that returns results including -Search, -Add, -Update, -Random, or -FindAll. The [Database_Names] ... [/Database_Names] and [Database_TableNames] ... [/Database_TableNames] tags can be used on their own.

Table 9: Schema Tags

Tag	Description
[Database_Names]	Container tag repeats for every database available to Lasso. Requires internal [Database_Nameltem] tag to show results.
[Database_Nameltem]	When used inside [Database_Names] ... [/Database_Names] container tags returns the name of the current database.
[Database_RealName]	Returns the real name of a database given an alias.
[Database_TableNames]	Container tag repeats for every table within a database. Accepts one required parameter, the name of the database. Requires internal [Database_Nameltem] tag to show results. Synonym is [Database_LayoutNames].
[Database_TableNameltem]	When used inside [Database_TableNames] ... [/Database_TableNames] container tags returns the name of the current table. Synonym is [Database_LayoutNameltem].
[Field_Name]	Returns the name of a field in the current database and table. A number parameter returns the name of the field in that position within the current table. Other parameters are described below. Synonym is [Column_Name].
[Field_Names]	Returns an array containing all the field names in the current result set. This is the same data as returned by [Field_Name], but in a format more suitable for iterating or other data processing. Synonym is [Column_Names].
[Required_Field]	Returns the name of a required field. Requires one parameter which is the number of the field name to return or a -Count keyword to return the total number of required fields.
[Table_RealName]	Returns the real name of a table given an alias. Requires a -Database parameter which specifies the database in which the table or alias resides.

To list all the databases available to Lasso:

The following example shows how to list the names of all available databases using the [Database_Names] ... [/Database_Names] and [Database_Nameltem] tags.

```
[Database_Names]
  <br>[Loop_Count]: [Database_Nameltem]
[/Database_Name]
```

→
1: Contacts

2: Examples

3: Site

To list all the tables within a database:

The following example shows how to list the names of all the tables within a database using the [Database_TableNames] ... [/Database_TableNames] and [Database_TableNameItem] tags. The tables within the Site database are listed.

```
[Database_TableNames: 'Site']
  <br>[Loop_Count]: [Database_TableNameItem]
[/Database_TableName]

→ <br>1: _outgoingemail
   <br>2: _outgoingemailprefs
   <br>3: _schedule
   <br>4: _sessions
```

To list all the fields within a table:

The [Field_Name] tag accepts a number of optional parameters which allow information about the tags in the current table to be returned. These parameters are detailed in *Table 10: [Field_Name] Parameters*.

Table 10: [Field_Name] Parameters

Parameter	Description
Number	The position of the field name to be returned. Required unless -Count is specified.
-Count	Returns the number of fields in the current table.
-Type	Returns the type of the field rather than the name. Types include Text, Number, Image, Date/Time, Boolean, or Unknown. Requires that a number parameter be specified.
-Protection	Returns the protection status of the field rather than the name. Protection statuses include None or Read Only. Requires that a number parameter be specified.

To return information about the fields in a table:

The following example demonstrates how to return information about the fields in a table using the [Inline] ... [/Inline] tags to perform a -Show action. [Loop] ... [/Loop] tags loop through the number of fields in the table and the name, type, and protection status of each field is returned. The fields within the Contacts Web table are shown.

```
[Inline: -Show,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
[Loop: (Field_Name: -Count)]
```

```

    <br>[Loop_Count]: [Field_Name: (Loop_Count)]
    ([Field_Name: (Loop_Count), -Type], [Field_Name: (Loop_Count), -Protection])
  [/Loop]
[/Inline]
→ <br>1: Creation Date (Date, None)
   <br>2: ID (Number, Read Only)
   <br>3: First_Name (Text, None)
   <br>4: Last_Name (Text, None)

```

To list all the required fields within a table:

The [Required_Field] tag accepts a number of optional parameters which allow information about the tags in the current table to be returned. These parameters are detailed in *Table 11: [Required_Field] Parameters*.

Table 11: [Required_Field] Parameters

Parameter	Description
Number	The position of the field name to be returned. Required unless -Count is specified.
-Count	Returns the number of required fields in the current table.

The [Required_Field] substitution tag can be used to return a list of all required fields for the current action. A -Show action is used to retrieve the information from the database and then [Loop] ... [/Loop] tags are used to loop through all the required fields. In the example that follows the People table of the Contacts database has only one required field, the primary key field ID.

```

  [Inline: -Show,
    -Database='Contacts',
    -Table='People']
  [Loop: (Required_Field: -Count)]
    <br>[Required_Field: (Loop_Count)]
  [/Loop]
[/Inline]
→ <br>ID

```

SQL Statements

LDML 7 provides the ability to issue SQL statements directly to SQL-compliant data sources, including the built-in Lasso MySQL data source. SQL statements are specified within the [Inline] tag using the -SQL command

tag. Many third-party databases that support SQL statements also support the use of the `-SQL` command tag.

SQL inlines can be used as the primary method of database interaction in Lasso 7, or they can be used along side standard inline actions (e.g. `-Search`, `-Add`, `-Update`, `-Delete`) where a specific SQL function is desired that cannot be replicated using standard database commands.

SQL Language Note: Documentation of SQL itself is outside the realm of this manual. Please consult the documentation included with your data source for information on what SQL statements are supported by it.

FileMaker Note: The `-SQL` inline parameter is not supported for FileMaker data sources.

Table 12: SQL Inline Parameters

Tag	Description
<code>-SQL</code>	Issues one or more SQL command to a compatible data source. Multiple commands are delimited by a semicolon. When multiple commands are used, all will be executed, however only the last command issued will return results to the <code>[Inline] ... [[/Inline]]</code> tags..
<code>-Database</code>	A database in the data source in which to execute the SQL statement.
<code>-MaxRecords</code>	The maximum number of records to return. Optional, defaults to 50.
<code>-SkipRecords</code>	The offset into the found set at which to start returning records. Optional, defaults to 1.

The `-Database` parameter can be any database within the data source in which the SQL statement should be executed. The `-Database` parameter will only be used to determine the data source, all table references within the statement must include both a database name and a table name, e.g. `Contacts.People`. For example, to create a new database in Lasso MySQL, a `CREATE DATABASE` statement can be executed with `-Database` set to `Site`.

When referencing the name of a database and table in a SQL statement (e.g. `Contacts.People`), only the true file names of a database or table can be used as MySQL does not recognize Lasso aliases in a SQL command. LDML 7 contains two SQL helper tags that return the true file name of a SQL database or table, as shown in *Table 13: SQL Helper Tags*.

Table 13: -SQL Helper Tags

Tag	Description
[Database_RealName]	Returns the actual name of a database from an alias. Useful for determining the true name of a database for use with the -SQL tag.
[Table_RealName]	This tag returns the actual name of a table from an alias. Useful for determining the true name of a table for use with the -SQL tag.

To determine the true database and table name for a SQL statement:

Use the [Database_RealName] and [Table_RealName] tags. When using the -SQL tag to issue SQL statements to a MySQL host, only true database and tables may be used (bypassing the alias). The [Database_RealName] and [Table_RealName] tags can be used to automatically determine the true name of a database and table, allowing them to be used in a valid SQL statement.

```
[Var_Set:'Real_DB' = (Database_RealName:'Contacts_Alias')]
[Var_Set:'Real_TB' = (Table_RealName:'Contacts_Alias')]
[Inline: -Database='Contacts_Alias', -SQL='select * from ((Var:'Real_DB') + '.' + (Var:
'Real_TB'))]
```

Results from a SQL statement are returned in a record set within the [Inline] ... [/Inline] tags. The results can be read and displayed using the [Records] ... [/Records] container tags and the [Field] substitution tag. However, many SQL statements return a synthetic record set that does not correspond to the names of the fields of the table being operated upon. This is demonstrated in the examples that follow.

To issue a SQL statement:

Specify the SQL statement within [Inline] ... [/Inline] tags in a -SQL command tag.

- The following example calculates the results of a mathematical expression $1 + 2$ and returns the value as a [Field] value named Result. Note that even though this SQL statement does not reference a database, a -Database tag is still required so Lasso knows to which data source to send the statement.

```
[Inline: -Database='Example', -SQL='SELECT 1+2 AS Result']
<br>The result is: [Field: 'Result'].
[/Inline]
```

→
The result is 3.

- The following example calculates the results of several mathematical expressions and returns them as field values One, Two, and Three.

```
[Inline: -Database='Example',
-SQL='SELECT 1+2 AS One, sin(.5) AS Two, 5%2 AS Three']
<br>The results are: [Field: 'One'], [Field: 'Two'], and [Field: 'Three'].
[/Inline]
```

→
The results are 3, 0.579426, and 1.

- The following example calculates the results of several mathematical expressions using LDML and returns them as field values One, Two, and Three. It demonstrate how the results of LDML expressions and substitution tags can be used in a SQL statement.

```
[Inline: -Database='Example',
-SQL='SELECT ' + (1+2) + ' AS One, ' + (Math_Sin: .5) +
' AS Two, ' + (Math_Mod: 5, 2) + ' AS Three']
<br>The results are: [Field: 'One'], [Field: 'Two'], and [Field: 'Three'].
[/Inline]
```

→
The results are 3, 0.579426, and 1.

- The following example returns records from the Phone_Book table where First_Name is equal to John. This is equivalent to a -Search using LDML.

```
[Inline: -Database='Example',
-SQL='SELECT * FROM Phone_Book WHERE First_Name = "John"]
[Records]
<br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

→
John Doe

John Person

To issue a SQL statement with multiple commands:

Specify the SQL statements within [Inline] ... [/Inline] tags in a -SQL command tag, with each SQL command separated by a semi-colon. The following example adds three unique records to the Contacts database. Note that all single quotes within the SQL statement have been properly escaped using the \ character, as described at the beginning of this chapter.

```
[Inline: -Database='Contacts',
-SQL='INSERT INTO Contacts.People (First_Name, Last_Name) VALUES
(\John\, \Jakob\);
INSERT INTO Contacts.People (First_Name, Last_Name) VALUES
(\Tom\, \Smith\);
INSERT INTO Contacts.People (First_Name, Last_Name) VALUES
(\Sally\, \Brown\)']
[/Inline]
```


To automatically format the results of a SQL statement:

Use the [Field_Name] tag and [Loop] ... [/Loop] tags to create an HTML table that automatically formats the results of a -SQL command. The -MaxRecords tag should be set to All so all records are returned rather than the default (50).

The following example shows a REPAIR TABLE Contacts.People SQL statement being issued to a MySQL database, and the result is automatically formatted. The statement returns a synthetic record set which shows the results of the repair.

Notice that the database Contacts is specified explicitly within the SQL statement. Even though the database is identified in the -Database command tag within the [Inline] tag it still must be explicitly specified in each table reference within the SQL statement.

```
[Inline: -Database='Contacts',
  -SQL='REPAIR TABLE Contacts.People',
  -MaxRecords='All']
<table border="1">
  <tr>
    [Loop: (Field_Name: -Count)]
    <td><b>[Field_Name: (Loop_Count)]</b></td>
  [/Loop]
</tr>
[Records]
  <tr>
    [Loop: (Field_Name: -Count)]
    <td>[Field: (Field_Name: Loop_Count)]</td>
  [/Loop]
</tr>
[/Records]
</table>
[/Inline]
```

The results are returned in a table with bold column headings. The following results show that the table did not require any repairs. If repairs are performed then many records will be returned.

→ Table	Op	Msg_Type	Msg_Text
People	Check	Status	OK

SQL Transactions

LDML 7 supports the ability to perform reversible SQL transactions provided that the data source used (e.g. MySQL 4.x) supports this func-

tionality. See your data source documentation to see if transactions are supported.

FileMaker Note: SQL transactions are not supported for FileMaker Pro data sources.

SQL transactions can be achieved within nested [Inline] ... [/Inline] tags. A single connection to MySQL or JDBC data sources will be held open from the opening [Inline] tag to the closing [/Inline] tag. Any nested inlines that use the same data source will make use of the same connection.

Note: When using named inlines, the connection is not available in subsequent [Records: -InlineName='Name'] ... [/Records] tags.

To open a transaction and commit or rollback in MySQL:

Used nested -SQL inlines, where the outer inline performs a transaction, and the inner inline commits or rolls back the transaction depending on the results of a conditional statement.

```
[Inline: -Database='Contacts', -SQL='START TRANSACTION
    INSERT INTO Contacts.People (Title, Company) VALUES ('Mr.', 'Blue World');]
[If: (Error_CurrentError) != (Error_NoError)]
    [Inline: -Database='Contacts', -SQL='ROLLBACK;']
[/Inline]
[Else]
    [Inline: -Database='Contacts', -SQL='COMMIT;']
[/Inline]
[/If]
[/Inline]
```

To fetch the last inserted ID in MySQL:

Used nested -SQL inlines, where the outer inline performs an insert query, and the inner inline retrieves the ID of the last inserted record using the MySQL last_insert_id() function. Because the two inlines share the same connection, the inner inline will always return the value added by the outer inline.

```
[Inline: -Database='Contacts',
    -SQL='INSERT INTO People (Title, Company) VALUES ('Mr.', 'Blue World');]
[Inline: -SQL='SELECT last_insert_id()']
    [Field: 'last_insert_id()']
[/Inline]
[/Inline]
```

→ 23

7

Chapter 7

Searching and Displaying Data

This chapter documents the LDML command tags which search for records and data within Lasso compatible databases and display the results.

- *Overview* provides an introduction to the database actions described in this chapter and presents important security considerations.
- *Searching Records* includes instructions for searching records within a database.
- *Displaying Data* describes the tags that can be used to display data that result from database searches.
- *Linking to Data* includes requirements and instructions for navigating through found sets and linking to particular records within a database.

Overview

LDML provides command tags for searching records within Lasso compatible databases. These command tags are used in conjunction with additional command tags and name/value parameters in order to perform the desired database action in a specific database and table or within a specific record.

The command tags documented in this chapter are listed in *Table 1: Command Tags*. The sections that follow describe the additional command tags and name/value parameters required for each database action.

Table 1: Command Tags

Tag	Description
-Search	Searches for records within a database.
-FindAll	Finds all records within a database.
-Random	Returns a random record from a database. Only works with FileMaker Pro databases.

How Searches are Performed

This section describes the steps that take place each time a search is performed using Lasso.

- 1 Lasso checks the database, table, and field name specified in the search to ensure that they are all valid.
- 2 Lasso security is checked to ensure that the current user has permission to perform a search in the desired database, table, and field. Filters are applied to the search criteria if they are defined within Lasso Administration.
- 3 The search query is formatted and sent to the database application. FileMaker Pro search queries are formatted as URLs and submitted to the Web Companion. Lasso MySQL search queries are formatted as SQL statements and submitted directly to Lasso MySQL.
- 4 The database application performs the desired search and assembles a found set. The database application is responsible for interpreting search criteria, wild cards in search strings, field operators, and logical operators.
- 5 The database application sorts the found set based on sort criteria included in the search query. The database application is responsible for determining the order of records returned to Lasso.
- 6 A subset of the found set is sent to Lasso as the result set. Only the number of records specified by `-MaxRecords` starting at the offset specified by `-SkipRecords` are returned to Lasso. If any `-ReturnField` command tags are included in a search then only those fields named by the `-ReturnField` command tags are returned to Lasso.
- 7 The result set can be displayed and manipulated using LDML tags that return information about the result set and LDML tags that return fields or other values.

Character Encoding

Lasso stores and retrieves data from data sources based on the preferences established in the **Setup > Data Sources** section of Lasso Administration. The following rules apply for each standard data source.

Lasso MySQL and MySQL – By default all communication is in the Latin-1 (ISO 8859-1) character set. This is to preserve backwards compatibility with prior versions of Lasso. The character set can be changed to the Unicode standard UTF-8 character set in the **Setup > Data Sources > Tables** section of Lasso Administration.

FileMaker Pro – By default all communication is in the MacRoman character set when Lasso Professional is hosted on Mac OS X or in the Latin-1 (ISO 8859-1) character set when Lasso Professional is hosted on Windows. The preference in the **Setup > Data Sources > Databases** section of Lasso Administration can be used to change the character set for cross-platform communications.

JDBC – All communication with JDBC data sources is in the Unicode standard UTF-8 character set.

See the Lasso Professional 7 Setup Guide for more information about how to change the character set settings in Lasso Administration.

Error Reporting

After a database action has been performed, Lasso reports any errors which occurred via the `[Error_CurrentError]` tag. The value of this tag should be checked to ensure that the database action was successfully performed.

To display the current error code and message:

The following code can be used to display the current error message. This code should be placed in a format file which is a response to a database action or within a pair of `[Inline] ... [/Inline]` tags.

```
[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
```

If the database action was performed successfully then the following result will be returned.

→ 0: No Error

To check for a specific error code and message:

The following example shows how to perform code to correct or report a specific error if one occurs. The following example uses a conditional `[If] ... [/If]` tag to check the current error message and see if it is equal to `[Error_NoRecordsFound]`.

```

[If: (Error_CurrentError) == (Error_NoRecordsFound)]
  No records were found!
[/If]

```

Full documentation about error tags and error codes can be found in *Chapter 21: Error Control*. A list of all Lasso error codes and messages can be found in *Appendix B: Error Codes*.

Classic Lasso

If Classic Lasso support has been disabled within Lasso Administration then database actions will not be performed automatically if they are specified within HTML forms or URLs. Although the database action will not be performed, the `-Response` tag will function normally. Use the following code in the response page to the HTML forms or URL to trigger the database action.

```

[Inline: (Action_Params)]
  [Error_CurrentError: -ErrorCode]: [Error_CurrentError]
[/Inline]

```

See *Chapter 6: Database Interaction Fundamentals* in this Lasso 7 Language Guide and *Chapter 6: Setting Global Preferences* in the Lasso Professional 7 Setup Guide for more information.

Note: The use of Classic Lasso has been deprecated. All solutions should be transitioned over to the `[Inline] ... [/Inline]` tag based methods described in this chapter.

Security

Lasso has a robust internal security system that can be used to restrict access to database actions or to allow only specific users to perform database actions. If a database action is attempted when the current visitor has insufficient permissions then they will be prompted for a username and password. An error will be returned if the visitor does not enter a valid username and password.

An `[Inline] ... [/Inline]` can be specified to execute with the permissions of a specific user by specifying `-Username` and `-Password` command tags within the `[Inline]` tag. This allows the database action to be performed even though the current site visitor does not necessarily have permissions to perform the database action. In essence, a valid username and password are embedded into the format file.

Table 2: Security Command Tags

Tag	Description
-Username	Specifies the username from Lasso Security which should be used to execute the database action.
-Password	Specifies the password which corresponds to the username.

To specify a username and password in an [Inline]:

The following example shows a -FindAll action performed within an [Inline] tag using the permissions granted for username SiteAdmin with password Secret.

```
[Inline: -FindAll,
  -Database='Contacts',
  -Table='People',
  -Username='SiteAdmin',
  -Password='Secret']

[Error_CurrentError: -ErrorCode]: [Error_CurrentError]

[/Inline]
```

A specified username and password is only valid for the [Inline] ... [/Inline] tags in which it is specified. It is not valid within any nested [Inline] ... [/Inline] tags. See *Chapter 8: Setting Up Security* of the Lasso Professional 7 Setup Guide for additional important information regarding embedding usernames and passwords into [Inline] tags.

Searching Records

Searches can be performed within any Lasso compatible database using the -Search command tag. The -Search command tag is specified within [Inline] ... [/Inline] tags. The -Search command tag requires that a number of additional command tags be defined in order to perform the search. The required command tags are detailed in *Table 3: -Search Action Requirements*.

Note: If Classic Lasso syntax is enabled then the -Search command tag can also be used within HTML forms or URLs. The use of Classic Lasso syntax has been deprecated so solutions which rely on it should be updated to use the inline methods described in this chapter.

Additional command tags are described in *Table 4: Operator Command Tags* and *Table 6: Results Command Tags* in the sections that follow.

Table 3: -Search Action Requirements

Tag	Description
-Search	The action which is to be performed. Required.
-Database	The database which should be searched. Required.
-Table	The table from the specified database which should be searched. Required.
-KeyField	The name of the field which holds the primary key for the specified table. Recommended.
-KeyValue	The particular value for the primary key of the record which should be returned. Using -KeyValue overrides all the other search parameters and returns the single record specified. Optional.
Name/Value Parameters	A variable number of name/value parameters specify the query which will be performed.

Any name/value parameters included in the search action will be used to define the query that is performed in the specified table. All name/value parameters must reference a field within the database. Any fields which are not referenced will be ignored for the purposes of the search.

To search a database using [Inline] ... [/Inline] tags:

The following example shows how to search a database by specifying the required command tags within an opening [Inline] tag. -Database is set to Contacts, -Table is set to People, and -KeyField is set to ID. The search returns records which contain John with the field First_Name.

The results of the search are displayed to the visitor inside the [Inline] ... [/Inline] tags. The tags inside the [Records] ... [/Records] tags will repeat for each record in the found set. The [Field] tags will display the value for the specified field from the current record being shown.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'='John']

[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]

[/Inline]
```

If the search was successful then the following results will be returned.

→
John Person

John Doe

Additional name/value parameters and command tags can be used to generate more complex searches. These techniques are documented in the following section on *Operators*.

To search a database using visitor-defined values:

The following example shows how to search a database by specifying the required command tags within an opening [Inline] tag, but allow a site visitor to specify the search criteria in an HTML form.

The visitor is presented with an HTML form in the format file default.lasso. The HTML form contains two text inputs for First_Name and Last_Name and a submit button. The action of the form is the response page response.lasso which contains the [Inline] ... [/Inline] tags that will perform the search. The contents of the default.lasso file include the following.

```
<form action="response.lasso" method="POST">
  <br>First Name: <input type="text" name="First_Name" value="">
  <br>Last Name: <input type="text" name="Last_Name" value="">
  <br><input type="submit" name="-Nothing" value="Search Database">
</form>
```

The search is performed and the results of the search are displayed to the visitor inside the [Inline] ... [/Inline] tags in response.lasso. The values entered by the visitor in the HTML form in default.lasso are inserted into the [Inline] tag using the [Action_Param] tag. The tags inside the [Records] ... [/Records] tags will repeat for each record in the found set. The [Field] tags will display the value for the specified field from the current record being shown. The contents of the response.lasso file include the following.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'=(Action_Param: 'First_Name'),
  'Last_Name'=(Action_Param: 'Last_Name')]

[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]

[/Inline]
```

If the visitor entered John for First_Name and Person for Last_Name then the following result would be returned.

→
John Person

Operators

LDML includes a set of command tags that allow operators to be used to create complex database queries. These command tags are summarized in *Table 4: Operator Command Tags*.

Table 4: Operator Command Tags

Tag	Description
-OperatorLogical	Specifies the logical operator for the search. Abbreviation is -OpLogical. Defaults to and.
-Operator	When specified before a name/value parameter, establishes the search operator for that name/value parameter. Abbreviation is -Op. Defaults to bw.
-OperatorBegin	Specifies the logical operator for all search parameters until -OperatorEnd is reached. Abbreviation is -OpBegin.
-OperatorEnd	Specifies the end of a logical operator grouping started with -OperatorBegin. Abbreviation is -OpEnd.

The operator command tags are divided into two categories.

- **Field Operators** are specified using the -Operator command tag before a name/value parameter. The field operator changes the way that the named field is searched for the value. If no field operator is specified then the default begins with bw operator is used. See *Table 5: Field Operators* for a list of the possible values for this tag.
- **Logical Operators** are specified using the -OperatorLogical, -OperatorBegin, and -OperatorEnd tags. These tags specify how the results of different name/value parameters are combined to form the full results of the search.

Field Operators

The possible values for the -Operator command tag are listed in *Table 5: Field Operators*. The default operator is begins with bw. Each operator can be used in its short form cn or in its long form Contains. Case is unimportant when specifying operators.

Field operators are interpreted differently depending on which database application is being accessed. For example, FileMaker Pro interprets bw to mean that any word within a field can begin with the value specified for that field. MySQL interprets bw to mean that the first word within the field must begin with the value specified. See the chapters on each data source or the documentation that came with a third-party data source connector for more information.

Several of the field operators are only supported in Lasso MySQL or other MySQL databases. These include the `ft` full text operator and the `rx` regular expression operators.

Table 5: Field Operators

Operator	Description
<code>bw</code>	Begins With. Default if no operator is set.
<code>cn</code>	Contains.
<code>ew</code>	Ends With.
<code>eq</code>	Equals.
<code>ft</code>	Full Text. MySQL databases only.
<code>gt</code>	Greater Than.
<code>gte</code>	Greater Than or Equals.
<code>lt</code>	Less Than.
<code>lte</code>	Less Than or Equals.
<code>neq</code>	Not Equals.
<code>nrx</code>	Not RegExp. Opposite of RegExp. MySQL databases only.
<code>rx</code>	RegExp. Regular expression search. MySQL databases only.

Note: In previous versions of Lasso the field operators could be specified using either a short form, e.g. `bw` or a long form, e.g. `Begins With`. In Lasso Professional 7 only the short form is preferred. Use of the long form is deprecated. It is supported in this version, but may not work in future versions of Lasso Professional.

To specify a field operator in an [Inline] tag:

Specify the field operator before the name/value parameter which it will affect. The following `[Inline] ... [/Inline]` tags search for records where the `First_Name` begins with `J` and the `Last_Name` ends with `son`.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -Operator='bw', 'First_Name'='J',
  -Operator='ew', 'Last_Name'='son']
```

```
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

The results of the search would include the following records.

```
→ <br>John Person
   <br>Jane Person
```

Logical Operators

The logical operator command tag `-OperatorLogical` can be used with a value of either `AND` or `OR`. The command tags `-OperatorBegin`, and `-OperatorEnd` can be used with values of `AND`, `OR`, or `NOT`. `-OperatorLogical` applies to all search parameters specified with an action `.` `-OperatorBegin` applies to all search parameters until the matching `-OperatorEnd` tag is reached. The case of the value is unimportant when specifying a logical operator.

- **AND** specifies that records which are returned should fulfill all of the search parameters listed.
- **OR** specifies that records which are returned should fulfill one or more of the search parameters listed.
- **NOT** specifies that records which match the search criteria contained between the `-OperatorBegin` and `-OperatorEnd` tags should be omitted from the found set. `NOT` cannot be used with the `-OperatorLogical` tag.

Note: In lieu of a `NOT` option for `-OperatorLogical`, many field operators can be negated individually by substituting the opposite field operator. The following pairs of field operators are the opposites of each other: `eq` and `neq`, `lt` and `gte`, `gt` and `lte`.

FileMaker Note: The `-OperatorBegin` and `-OperatorEnd` tags do not work with Lasso Connector for FileMaker Pro.

To perform a search using an AND operator:

Use the `-OperatorLogical` command tag with an `AND` value. The following `[Inline] ... [Inline]` tags return records for which the `First_Name` field begins with John and the `Last_Name` field begins with Doe. The position of the `-OperatorLogical` command tag within the `[Inline]` tag is unimportant since it applies to the entire action.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -OperatorLogical='AND',
```

```
'First_Name'='John',
'Last_Name'='Doe']
[Records]
<br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

To perform a search using an OR operator:

Use the -OperatorLogical command tag with an OR value. The following [Inline] ... [/Inline] tags return records for which the First_Name field begins with either John or Jane. The position of the -OperatorLogical command tag within the [Inline] tag is unimportant since it applies to the entire action.

```
[Inline: -Search,
-Database='Contacts',
-Table='People',
-KeyField='ID',
-OperatorLogical='OR',
'First_Name'='John',
'First_Name'='Jane']
[Records]
<br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

To perform a search using a NOT operator:

Use the -OperatorBegin and -OperatorEnd command tags with a NOT value. The following [Inline] ... [/Inline] tags return records for which the First_Name field begins with John and the Last_Name field is not Doe. The operators tags must surround the parameters of the search which are to be negated.

```
[Inline: -Search,
-Database='Contacts',
-Table='People',
-KeyField='ID',
'First_Name'='John',
-OperatorBegin='NOT',
'Last_Name'='Doe',
-OperatorEnd='NOT']
```

```
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

To perform a search with a complex query:

Use the -OperatorBegin and -OperatorEnd tags to build up a complex query. As an example, a query can be constructed to find records in a database whose First_Name and Last_Name both begin with the same letter J, or M. The desired query could be written in pseudo-code as follows.

```
( (First_Name begins with J) AND (Last_Name begins with J) ) OR
( (First_Name begins with M) AND (Last_Name begins with M) )
```

The pseudo code is translated into a URL as follows. Each line of the query becomes a pair of -OpBegin=AND and -OpEnd=AND tags with a name/value parameter for First_Name and Last_Name contained inside. The two lines are then combined using a pair of -OpBegin=OR and -OpEnd=OR tags. The nesting of the command tags works like the nesting of parentheses in the pseudo code above to clarify how Lasso should combine the results of different name/value parameters.

```
<a href="/response.lasso?-Search&
-Database=Contacts&
-Table=People&
-KeyField=ID&
-OpBegin=OR&
  -OpBegin=AND&
    First_Name=J&
    Last_Name=J&
  -OpEnd=AND&
  -OpBegin=AND&
    First_Name=M&
    Last_Name=M&
  -OpEnd=AND&
-OpEnd=OR">
  First Name and Last Name both begin with J or M
</a>
```

The following results might be returned when this link is selected.

```
→ <br>Johnny Johnson
   <br>Jimmy James
   <br>Mark McPerson
```

Results

LDML includes a set of command tags that allow the results of a search to be customized. These command tags do not change the found set of records that are returned from the search, but they do change the data that is returned to Lasso for formatting and display to the visitor. The results command tags are summarized in *Table 6: Results Command Tags*.

Table 6: Results Command Tags

Tag	Description
-Distinct	Specifies that only records with distinct values in all returned fields should be returned. MySQL databases only.
-MaxRecords	Specifies how many records should be shown from the found set. Optional, defaults to 50.
-SkipRecords	Specifies an offset into the found set at which records should start being shown. Optional, defaults to 1.
-ReturnField	Specifies a field that should be returned in the results of the search. Multiple -ReturnField tags can be used to return multiple fields. Optional, defaults to returning all fields in the searched table.
-SortField	Specifies that the results should be searched based on the data in the named field. Multiple -SortField tags can be used for complex sorts. Optional, defaults to returning data in the order it appears in the database.
-SortOrder	When specified after a -SortField parameter, specifies the order of the sort, either ascending, descending or custom. Optional, defaults to ascending for each -SortField.
-SortRandom	Sorts the returned results randomly. MySQL databases only.
-UseLimit	Specifies that a MySQL LIMIT should be used instead of Lasso's built-in tools for limiting the found set. MySQL databases only.

The results command tags are divided into three categories.

- **Sorting** is specified using the -SortField and -SortOrder command tags. These tags change the order of the records which are returned by the search. The sort is performed by the database application before Lasso receives the record set.

The -SortRandom tag can be used to perform a random sort on the found set from MySQL databases. Note that the sort will be random each time

a set of records is returned so `-MaxRecords` and `-SkipRecords` cannot be used to navigate a found set that is sorted randomly.

- The portion of the **Found Set** being shown is specified using the `-MaxRecords` and `-SkipRecords` tags. `-MaxRecords` sets the number of records which will be shown between the `[Records]` ... `[/Records]` tags that format the results for the visitor. The `-SkipRecords` tag sets the offset into the found set which is shown. These two tags define the window of records which are shown and can be used to navigate through a found set.

The `-UseLimit` tag instructs MySQL data sources to use a SQL LIMIT tag to restrict the found set based on the values of the `-MaxRecords` and `-SkipRecords` tags. This may increase performance when many records are being found, but `-MaxRecords` is set to a low value.

- The **Fields** which are available are specified using the `-ReturnField` tag. Normally, all fields in the table that was searched are returned. If any `-ReturnField` tags are specified then only those fields will be available to be returned to the visitor using the `[Field]` tag. Specifying `-ReturnField` tags can improve the performance of Lasso by not sending unnecessary data between the database and the Web server.

Note: In order to use the `[KeyField_Value]` tag within an inline the keyfield must be specified as one of the `-ReturnField` values.

- The `-Distinct` tag instructs MySQL data sources to return only records which contain distinct values across all returned fields. This tag is useful when combined with a single `-ReturnField` tag and a `-FindAll` to return all distinct values from a single field in the database.

To return sorted results:

Specify `-SortField` and `-SortOrder` command tags within the search parameters. The following inline includes sort command tags. The records are first sorted by `Last_Name` in ascending order, then sorted by `First_Name` in ascending order.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'='J',
  -SortField='Last_Name', -SortOrder='Ascending',
  -SortField='First_Name', -SortOrder='Descending']
```



```
[Records]
  <br>[Field: 'First_Name']
[/Records]
[/Inline]
```

The following results could be returned when this inline is run. The returned records are sorted in order of Last_Name. If the Last_Name of two records are equal then those records are sorted in order of First_Name.

```
→ <br>Jane Doe
   <br>John Doe
   <br>Jane Person
   <br>John Person
```

To return a portion of a found set:

A portion of a found set can be returned by manipulating the values for -MaxRecords and -SkipRecords. In the following example, a search is performed for records where the First_Name begins with J. This search returns four records, but only the second two records are shown.

-MaxRecords is set to 2 to show only two records and -SkipRecords is set to 2 to skip the first two records.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'='J',
  -MaxRecords=2,
  -SkipRecords=2]
[Records]
  <br>[Field: 'First_Name']
[/Records]
[/Inline]
```

The following results could be returned when this inline is run. Neither of the Doe records from the previous example are shown since they are skipped over.

```
→ <br>Jane Person
   <br>John Person
```

To limit the fields returned in search results:

Use the -ReturnField command tag. If a single -ReturnField command tag is used then only the fields that are specified will be returned. If no -ReturnField command tags are specified then all fields within the current table will be shown. In the following example, only the First_Name field is shown since it is the only field specified within a -ReturnField command tag.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'='J',
  -ReturnField='First_Name']
[Records]
  <br>[Field: 'First_Name']
[/Records]
[/Inline]
```

The following results could be returned when this link is selected. The Last_Name field cannot be shown for any of these records since it was not specified in a -ReturnField command tag.

```
→ <br>Jane
   <br>John
   <br>Jane
   <br>John
```

If [Field: 'Last_Name'] were specified inside the [Inline] ... [/Inline] tags and not specified as a -ReturnField then an error would be returned rather than the indicated results.

Finding All Records

All records can be returned from a database using the -FindAll command tag. The -FindAll command tag functions exactly like the -Search command tag except that no name/value parameters or operator tags are required. Sort tags and tags which sort and limit the found set work the same as they do for -Search actions. -FindAll actions can be specified in [Inline] ... [/Inline] tags.

Note: If Classic Lasso syntax is enabled then the -FindAll command tag can also be used within HTML forms or URLs. The use of Classic Lasso syntax has been deprecated so solutions which rely on it should be updated to use the inline methods described in this chapter.

Table 7: -FindAll Action Requirements

Tag	Description
-FindAll	The action which is to be performed. Required.
-Database	The database which should be searched. Required.
-Table	The table from the specified database which should be searched. Required.
-KeyField	The name of the field which holds the primary key for the specified table. Recommended.

To find all records within a database:

The following [Inline] ... [/Inline] tags find all records within a database Contacts and displays them. The results are shown below.

```
[Inline: -FindAll,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
→ <br>Jane Doe
   <br>John Person
   <br>Jane Person
   <br>John Doe
```

To return all unique field values:

The unique values from a field in a MySQL database can be returned using the -Distinct tag. Only records which have distinct values across all fields will be returned. In the following example, a -FindAll action is used on the People table of the Contacts database. Only distinct values from the Last_Name field are returned.

```
[Inline: -FindAll,
  -Database='Contacts',
  -Table='People',
  -Distinct,
  -SortField='First_Name',
  -ReturnField='First_Name']
```

```
[Records]
  <br>[Field: 'First_Name']
[/Records]
[/Inline]
```

The following results are returned. Even though there are multiple instances of John and Jane in the database, only one record for each name is returned.

→
Jane

John

Finding Random Records

A random record can be returned from a database using the -Random command tag. The -Random command tag functions exactly like the -Search command tag except that no name/value parameters or operator tags are required. -Random actions can be specified in [Inline] ... [/Inline] tags.

Note: If Classic Lasso syntax is enabled then the -Random command tag can also be used within HTML forms or URLs. The use of Classic Lasso syntax has been deprecated so solutions which rely on it should be updated to use the inline methods described in this chapter.

Table 8: -Random Action Requirements

Tag	Description
-Random	The action which is to be performed. Required.
-Database	The database which should be searched. Required.
-Table	The table from the specified database which should be searched. Required.
-KeyField	The name of the field which holds the primary key for the specified table. Recommended.

To find a single random record from a database:

The following inline finds a single random record from a FileMaker Pro database Contacts.fp3 and displays it. -MaxRecords is set to 1 to ensure that only a single record is shown. One potential result is shown below. Each time this inline is run a different record will be returned.

```
[Inline: -Random,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords=1]
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
→<br>Jane Person
```

To return multiple records sorted in random order:

The `-SortRandom` tag can be used with the `-Search` or `-FindAll` actions to return many records from a MySQL database sorted in random order. In the following example, all records from the `People` table of the `Contacts` database are returned in random order.

```
[Inline: -FindAll,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -SortRandom]
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
→ <br>John Doe
   <br>Jane Doe
   <br>Jane Person
   <br>John Person
```

Displaying Data

The examples in this chapter have all relied on the `[Records] ... [/Records]` tags and `[Field]` tag to display the results of the search that have been performed. This section describes the use of these tags in more detail.

Table 9: Field Display Tags

Tag	Description
[Records] ... [/Records]	Loops through each record in a found set. Optional -InlineName parameter specifies that results should be returned from a named inline. Synonym is [Rows] ... [/Rows].
[Field]	Returns the value for a database field. Requires one parameter, the field name. Optional parameter -RecordIndex specifies what record in the current found set a field should be shown from. Synonym is [Column].

The [Field] tag always returns the value for a field from the current record when it is used within [Records] ... [/Records] tags. If the [Field] tag is used outside of [Records] ... [/Records] tags then it returns the value for a field from the first record in the found set. If the found set is only one record then the [Records] ... [/Records] tags are optional.

FileMaker Note: Lasso Connector for FileMaker Pro includes a collection of FileMaker Pro specific tags which return database results. See *Chapter 10: FileMaker Pro Data Sources* for more information.

To display the results from a search:

Use the [Records] ... [/Records] tags and [Field] tag to display the results of a search. The following [Inline] ... [/Inline] tags perform a -FindAll action in a database Contacts. The results are returned each formatted on a line by itself. The [Loop_Count] tag is used to indicate the order within the found set.

```
[Inline: -FindAll,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
[Records]
  <br>[Loop_Count]: [Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]

→ <br>1: Jane Doe
   <br>2: John Person
   <br>3: Jane Person
   <br>4: John Doe
```

To display the results for a single record:

Use [Field] tags within the contents of the [Inline] ... [/Inline] tags. The [Records] ... [/Records] tags are unnecessary if only a single record is returned.

The following [Inline] ... [/Inline] tags perform a -Search for a single record whose primary key ID equals 1. The [KeyField_Value] is shown along with the [Field] values for the record.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -KeyValue=1]
<br>[KeyField_Value]: [Field: 'First_Name'] [Field: 'Last_Name']
[/Inline]

→ <br>1: Jane Doe
```

To display the results from a named inline:

Use the -InlineName parameter in both the opening [Inline] tag and in the opening [Records] tag. The [Records] ... [/Records] tags can be located anywhere in the page after the [Inline] ... [/Inline] tags that define the database action. The following example shows a -FindAll action at the top of a page in a LassoScript with the results formatted later.

```
<?LassoScript
  Inline: -FindAll,
    -Database='Contacts',
    -Table='People',
    -KeyField='ID',
    -InlineName='FindAll Results';
  /Inline;
?>

... Page Contents ...

[Records: -InlineName='FindAll Results']
  <br>[Loop_Count]: [Field: 'First_Name'] [Field: 'Last_Name']
[/Records]

→ <br>1: Jane Doe
   <br>2: John Person
   <br>3: Jane Person
   <br>4: John Doe
```

To display the results from a search out of order:

The -RecordIndex parameter of the [Field] tag can be used to show results out of order. Instead of using [Records] ... [/Records] tags to loop through a found set, the following example uses [Loop] ... [/Loop] tags to loop down through the found set from [MaxRecords_Value] to 1. The [Field] tags all reference the [Loop_Count] in their -RecordIndex parameter.

```
[Inline: -FindAll,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
[Loop: -LoopFrom=(MaxRecords_Value), -LoopTo=1, -LoopIncrement=-1]
  <br>[Loop_Count]: [Field: 'First_Name', -RecordIndex=(Loop_Count)]
  [Field: 'Last_Name', -RecordIndex=(Loop_Count)]
[/Loop]
[/Inline]
→ <br>4: John Doe
   <br>3: Jane Person
   <br>2: John Person
   <br>1: Jane Doe
```

Linking to Data

This section describes how to create links which allow a visitor to manipulate the found set. The following types of links can be created.

- **Navigation** – Links can be created which allow a visitor to page through a found set. Only a portion of the found set needs to be shown, but the entire found set can be accessed.
- **Detail** – Links can be created which allow detail about a particular record to be shown in another format file.
- **Sorting** – Links can be provided to re-sort the current found set on a different field.

Note: If Classic Lasso syntax is enabled then the links tags can be used to trigger actions using command tags embedded in URLs. The use of Classic Lasso syntax has been deprecated so solutions which rely on it should be updated to use the inline methods described in this chapter.

Most of the link techniques implicitly assume that the records within the database are not going to change while the visitor is navigating through the found set. The database search is actually performed again for every page served to a visitor and if the number of results change then the records being shown to the visitor can be shifted or altered as soon as another link is selected.

Link Tags

LDML 7 includes many tags which make creating detail links and navigation links easy within Lasso solutions. The general purpose link tags are specified in *Table 10: Link Tags*. The common parameters for all link tags are specified in *Table 11: Link Tag Parameters*.

The remainder of the chapter lists and demonstrates the link URL, container, and parameter tags. Tags which generate URLs for links automatically are listed in *Table 12: Link URL Tags*. Container tags which generate entire HTML anchor tags <a> automatically are listed in *Table 13: Link Container Tags*. Tags which provide parameter arrays for each link option are listed in *Table 14: Link Parameter Tags*.

Table 10: Link Tags

Tag	Description
[Link] ... [/Link]	General purpose link tag that provides an anchor tag with the specified parameters. The -Response parameter is used as the URL for the link.
[Link_Params]	General purpose link tag that processes a set of parameters using the common rules for all link tags.
[Link_SetFormat]	Sets a standard set of options that will be used for all link tags that follow in the current format file.
[Link_URL]	General purpose link tag that provides a URL based on the specified parameters. The -Response parameter is used as the URL for the link.

Each of the general purpose link tags implement the basic behavior of all the link tags, but are not usually used on their own. The section on *Link Tag Parameters* below describes the common parameters that all link tags interpret. The following sections include the link URL, container, and parameter tags and examples of their use.

Note: The [Link_...] tags do not include values for the -SQL, -Username, -Password or the -ReturnField tags in the links they generate.

Link Tag Parameters

All of the link tags accept the same parameters which allow the link that is being formed to be customized. These parameters include all the command tags which can be passed to the opening [Inline] tag and a series of parameters detailed in *Table 11: Link Tag Parameters* which allow various command tags to be removed from the generated link tags.

The link tags interpret their parameters as follows.

- The parameters are processed in the order they are specified within the link tag. Later parameters override earlier parameters.
- Most link tags process [Action_Params] first, then any parameters specified in [Link_SetFormat], and finally the parameters specified within the link tag itself. The general purpose link tags do not include [Action_Params] automatically.
- Parameters of type array are inserted into the parameters as if each item of the array was specified in order at the location of the array.
- Many command tags will only be included once in the resulting link. These include -Database, -Table, -KeyField, -MaxRecords, and any other command tags that can only be specified once within an inline. The last value for the command tag will be included in the resulting link.
- Only one action such as -Search, -FindAll, or -Nothing will be included in the resulting link. The last action specified in the link tag will be used.
- Command tags such as -Required, -Op, -OpBegin, -OpEnd, -SortField, -SortOrder, and -Token will be included in the order they are specified within the tag.
- The resulting link will consist of the action followed by all command tags specified once in alphabetical order, and finally all name/value parameters and command tags that are specified multiple times in the same order they were specified in the parameters.
- All -No... parameters are interpreted at the location they occur in the parameters. If a -NoDatabase parameter is specified early in the parameter list and a -Database command tag is included later then the -Database command tag will be included in the resulting link.
- The -NoClassic parameter removes all command tags that are not essential to specifying the search and location in the found set to an [Inline] tag. The -Database, -Table, -KeyField, and action are all removed. All name/value parameters, -Sort... tags, -Op tags, and either -MaxRecords and -SkipRecords or -KeyValue are included.
- The value of the -Response command tag will be used as the URL for the resulting link. The link tags always link to a response file on the same server they are called. If not specified the -Response will be the same as [Response_FilePath].
- The -SQL, -Username, -Password, and -ReturnField tags are never returned by the link tags.

Note: The [Referrer] and [Referrer_URL] tags are special cases which simply return the referrer specified in the HTTP request header. They do not accept any parameters.

Table 11: Link Tag Parameters

Tag	Description
Command Tag	Inserts the specified command tag. Either appends the command tag or overrides an existing command tag with the new value.
Name/Value Pair	Inserts the specified name/value pair.
Array Parameter	An array of pairs is inserted as if each name/value pair in the array was specified in the tag parameters at the location of the array.
-NoAction	Removes the action command tag.
-NoClassic	Removes all parameters required to specify an action in Classic Lasso leaving only those parameters required to specify the query and current location in the found set.
-NoDatabase	Removes the -Database command tag.
-NoTable	Removes the -Table or -Layout command tag. -NoLayout is a synonym.
-NoKeyField	Removes the -KeyField command tag.
-NoKeyValue	Removes the -KeyValue command tag.
-NoOperatorLogical	Removes the -OperatorLogical command tag.
-NoResponse	Removes the -Response command tag.
-NoMaxRecords	Removes the -MaxRecords command tag.
-NoSkipRecords	Removes the -SkipRecords command tag.
-NoParams	Removes name/value pairs, -Operator, -OperatorBegin, -OperatorEnd, and -Required tags.
-NoSort	Removes all -Sort... command tags.
-NoToken, -NoToken.Name	Removes the -Token command tag. With a parameter as -NoToken.Name removes the specified token command tag.
-NoTokens	Removes all -Token... command tags.
-NoSchema	Removes the -Schema command tag for JDBC data sources.
-No.Name	Removes a specified nam/value parameter.
-Response	Specifies the file that will be used as the URL for the link tag. The link tags always link to a file on the current server.

Link URL Tags

The tags listed in *Table 12: Link URL Tags* each return a URL based on the current database action. Each of these tags accepts the same parameters as specified in *Table 11: Link Tag Parameters* above and corresponds to matching container and parameter tags. Examples of the link tags are included in the *Link Examples* section that follows.

Table 12: Link URL Tags

Tag	Description
[Link_CurrentActionURL]	Returns a link to the current Lasso action.
[Link_FirstGroupURL]	Returns a link to the first group of records based on the current Lasso action. Sets -SkipRecords to 0.
[Link_PrevGroupURL]	Returns a link to the next group of records based on the current Lasso action. Changes -SkipRecords.
[Link_NextGroupURL]	Returns a link to the next group of records based on the current Lasso action. Changes -SkipRecords.
[Link_LastGroupURL]	Returns a link to the last group of records based on the current Lasso action. Changes -SkipRecords.
[Link_CurrentRecordURL]	Returns a link to the current record. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_FirstRecordURL]	Returns a link to the first record based on the current Lasso action. Sets -MaxRecords to 1 and -SkipRecords to 0.
[Link_PrevRecordURL]	Returns a link to the next record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_NextRecordURL]	Returns a link to the next record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_LastRecordURL]	Returns a link to the last record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_DetailURL]	Returns a link to the current record using the primary key and key value. Changes -KeyValue.
[Referrer_URL]	Returns a link to the previous page which the visitor was at before the current page. [Referer_URL] is a synonym.

Note: The [Referrer_URL] tag is a special case which simply returns the referrer specified in the HTTP request header. It does not accept any parameters.

Link Container Tags

The tags listed in *Table 13: Link Container Tags* each return an anchor tag based on the current database action. The anchor tags surround the contents of the container tag. If the link tag is not valid then no result is returned. Each of these tags accepts the same parameters as specified in *Table 11: Link Tag Parameters* above and corresponds to matching URL and parameter tags. Examples of the link tags are included in the *Link Examples* section that follows.

Table 13: Link Container Tags

Tag	Description
[Link_CurrentAction]	Returns a link to the current Lasso action.
[Link_FirstGroup]	Returns a link to the first group of records based on the current Lasso action. Sets -SkipRecords to 0.
[Link_PrevGroup]	Returns a link to the previous group of records based on the current Lasso action. Changes -SkipRecords.
[Link_NextGroup]	Returns a link to the next group of records based on the current Lasso action. Changes -SkipRecords.
[Link_LastGroup]	Returns a link to the last group of records based on the current Lasso action. Changes -SkipRecords.
[Link_CurrentRecord]	Returns a link to the current record. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_FirstRecord]	Returns a link to the first record based on the current Lasso action. Sets -MaxRecords to 1 and -SkipRecords to 0.
[Link_PrevRecord]	Returns a link to the previous record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_NextRecord]	Returns a link to the next record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_LastRecord]	Returns a link to the last record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_Detail]	Returns a link to the current record using the -KeyField and -KeyValue. Changes -KeyValue.
[Referrer]	Returns a link to the previous page which the visitor was at before the current page. [Referer] is a synonym.

Note: The [Referrer] ... [/Referrer] tag is a special case which simply returns the referrer specified in the HTTP request header. It does not accept any parameters.

Link Parameter Tags

The tags listed in *Table 14: Link Parameter Tags* each return an array of parameters based on the current database action. Each of these tags accepts the same parameters as specified in *Table 11: Link Tag Parameters* above and corresponds to matching container and URL tags. Examples of the link tags are included in the *Link Examples* section that follows.

Table 14: Link Parameter Tags

Tag	Description
[Link_CurrentActionParams]	Returns a link to the current Lasso action.
[Link_FirstGroupParams]	Returns a link to the first group of records based on the current Lasso action. Sets -SkipRecords to 0.
[Link_PrevGroupParams]	Returns a link to the next group of records based on the current Lasso action. Changes -SkipRecords.
[Link_NextGroupParams]	Returns a link to the next group of records based on the current Lasso action. Changes -SkipRecords.
[Link_LastGroupParams]	Returns a link to the last group of records based on the current Lasso action. Changes -SkipRecords.
[Link_CurrentRecordParams]	Returns a link to the current record. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_FirstRecordParams]	Returns a link to the first record based on the current Lasso action. Sets -MaxRecords to 1 and -SkipRecords to 0.
[Link_PrevRecordParams]	Returns a link to the next record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_NextRecordParams]	Returns a link to the next record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_LastRecordParams]	Returns a link to the last record based on the current Lasso action. Sets -MaxRecords to 1 and changes -SkipRecords.
[Link_DetailParams]	Returns a link to the current record using the primary key and key value. Changes -KeyValue.

Note: There is no link parameter tag equivalent to the referrer tags.

Link Examples

The basic technique for using the link tags is the same as that which was described to allow site visitors to enter values into HTML forms and then use those values within an `[Inline] ... [/Inline]` action. The `[Inline]` tags can have some command tags and search parameters specified explicitly, with variables, an array, `[Action_Params]`, or one of the link tags defining the rest.

For example, an `[Inline] ... [/Inline]` could be specified to find all records within a database as follows. The entire action is specified within the opening `[Inline]` tag. Each time a page with the code on it is visited the action will be performed as written.

```
[Inline: -FindAll,
      -Database='Contacts',
      -Table='People',
      -KeyField='ID',
      -MaxRecords=10]
...
[/Inline]
```

The same inline can be modified so that it can accept parameters from an HTML form or URL which is used to load the page it is on, but can still act as a standalone action. This is accomplished by adding an `[Action_Params]` tag to the opening `[Inline]` tag.

```
[Inline: (Action_Params),
      -Search,
      -Database='Contacts',
      -Table='People',
      -KeyField='ID',
      -MaxRecords=4]
...
[/Inline]
```

Any command tags or name/value pairs in the HTML form or URL that triggers the page with this inline will be passed into the inline through the `[Action_Params]` tag as if they had been typed directly into the `[Inline]`. However, the command tags specified directly in the `[Inline]` tag will override any corresponding tags from the `[Action_Params]`.

Since the action `-Search` is specified after the `[Action_Params]` array it will override any other action from the array. The action of this inline will always be `-Search`. Similarly, all of the `-Database`, `-Table`, `-KeyField`, or `-MaxRecords` tags will have the values specified in the `[Inline]` overriding any values passed in through `[Action_Params]`.

The various link tags can be used to generate URLs which work with the specified inline in order to change the set of records being shown, the sort order and sort field, etc. The link tags are able to override any command

tags not specified in the opening [Inline] tag, but the basic action is always performed exactly as specified.

Navigation Links

Navigation links are created by manipulating the value for -SkipRecords so that the visitor is shown a different portion of the found set each time they follow a link or by setting -KeyValue to an appropriate value to show one record in a database.

To create next and previous links:

The [Link_NextGroup] ... [/Link_NextGroup] and [Link_PrevGroup] ... [/Link_PrevGroup] tags can be used with the inline specified above to page through a set of found records.

The [Link_SetFormat] tag is used to include a -NoClassic parameter in each link tag that follows. This ensures that the -Database, -Table, and -KeyField are not included in the links generated by the link tags.

The full inline is shown below. It uses the [Records] ... [/Records] tags to show the people that have been found in the database and includes next and previous links to page through the found set.

```
[Inline: (Action_Params),
  -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords=4]

<p>[Found_Count] records were found, showing [Shown_Count]
records from [Shown_First] to [Shown_Last].

[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]

[Link_SetFormat: -NoClassic]
[Link_PrevGroup] <br>Previous [MaxRecords_Value] Records [/Link_PrevGroup]
[Link_NextGroup] <br>Next [MaxRecords_Value] Records [/Link_NextGroup]
[/Inline]
```

The first time this page is loaded the first four records from the database are shown. Since this is the first group of records in the database only the Next 4 Records link is displayed.


```
→ <p>16 records were found, showing 4 records from 1 to 4.
    <br>Jane Doe
    <br>John Person
    <br>Jane Person
    <br>John Doe
    <br>Next 4 Records
```

If the Next 4 Records link is selected then the same page is reloaded. The value for -SkipRecords is taken from the link tag and passed into the opening [Inline] tag through the [Action_Params] array. The following results are displayed. This time both the Next 4 Records and the Previous 4 Records links are displayed.

```
→ <p>16 records were found, showing 4 records from 5 to 8.
    <br>Jane Surname
    <br>John Last_Name
    <br>Mark Last_Name
    <br>Tom Surname
    <br>Previous 4 Records
    <br>Next 4 Records
```

To create first and last links:

Links to the first and last groups of records in the found set can be added using the [Link_FirstGroup] ... [/Link_FirstGroup] and [Link_LastGroup] ... [/Link_LastGroup] tags. The following inline includes both next/previous links and first/last links.

```
[Inline: (Action_Params),
  -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords=4]

<p>[Found_Count] records were found, showing [Shown_Count]
records from [Shown_First] to [Shown_Last].

[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]

[Link_SetFormat: -NoClassic]
[Link_FirstGroup] <br>First [MaxRecords_Value] Records [/Link_FirstGroup]
[Link_PrevGroup] <br>Previous [MaxRecords_Value] Records [/Link_PrevGroup]
[Link_NextGroup] <br>Next [MaxRecords_Value] Records [/Link_NextGroup]
[Link_LastGroup] <br>Last [MaxRecords_Value] Records [/Link_LastGroup]
[/Inline]
```

The first time this page is loaded the first four records from the database are shown. Since this is the first group of records in the data-

base only the Next 4 Records and Last 4 Records links are displayed. The Previous 4 Records and First 4 Records links will automatically appear if either of these links are selected by the visitor.

```
→ <p>16 records were found, showing 4 records from 1 to 4.
    <br>Jane Doe
    <br>John Person
    <br>Jane Person
    <br>John Doe
    <br>&u>Next 4 Records
    <br>&u>Last 4 Records
```

To create links to page through the found set:

Many Web sites include page links which allow the visitor to jump directly to any set of records within the found set. The example -FindAll returns 16 records from Contacts so four page links would be created to jump to the 1st, 5th, 9th, and 13th records.

A set of page links can be created using the [Link_CurrentActionURL] tag as a base and then customizing the -SkipRecords value as needed. The following loop creates as many page links as are needed for the current found set.

```
[Inline: (Action_Params),
  -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords=4]

<p>[Found_Count] records were found, showing [Shown_Count]
records from [Shown_First] to [Shown_Last].

[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]

[Link_SetFormat: -NoClassic]
[Variable: 'Count' = 0]
[While: $Count < (Found_Count)]
  <br><a href="[Link_CurrentActionURL: -SkipRecords=$Count]">
    Page [Loop_Count]
  </a>
  [Variable: 'Count' = $Count + (MaxRecords_Value)]
[/While]

[/Inline]
```

The results of this code for the example -Search would be the following. There are four page links. The first is equivalent to the First 4 Records link

created above and the last is equivalent to the Last 4 Records link created above.

```
→ <p>16 records were found, showing 4 records from 1 to 4.
    <br>Jane Doe
    <br>John Person
    <br>Jane Person
    <br>John Doe
    <br>Page 1
    <br>Page 2
    <br>Page 3
    <br>Page 4
```

Sorting Links

Sorting links are created by adding or manipulating `-SortField` and `-SortOrder` command tags. The same found set is shown, but the order is determined by which link is selected. Often, the column headers in a table of results from a database will represent the sort links that allow the table to be resorted by the values in that specific column.

To create links that sort the found set:

The following code performs a `-Search` in an inline and formats the results as a table. The column heading at the top of each table column is a link which re-sorts the results by the field values in that column. The links for sorting the found set are created by specifying `-NoSort` and `-SortField` parameters to the `[Link_FirstGroup] ... [/Link_FirstGroup]` tags.

```
[Inline: (Action_Params),
  -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords=4]

[Link_SetFormat: -NoClassic]
<table>
  <tr>
    <th>
      [Link_FirstGroup: -NoSort, -SortOrder='First_Name']
      First Name
    [/Link_FirstGroup]
    </th>
    <th>
      [Link_FirstGroup: -NoSort, -SortOrder='Last_Name']
```

```

                Last Name
            [/Link_FirstGroup]
        </th>
    </tr>

    [Records]
    <tr>
        <td>[Field: 'First_Name']</td>
        <td>[Field: 'Last_Name']</td>
    </tr>
    [/Records]

</table>
[/Inline]

```

Detail Links

Detail links are created in order to show data from a particular record in the database table. Usually, a listing format file will contain only limited data from each record in the found set and a detail format file will contain significantly more information about a particular record.

A link to a particular record can be created using the [Link_Detail] ... [/Link_Detail] tags to set the -KeyField and -KeyValue fields. This method is guaranteed to return the selected record even if the database is changing while the visitor is navigating. However, it is difficult to create next and previous links on the detail page. This option is most suitable if the selected database record will need to be updated or deleted.

Alternately, a link to a particular record can be created using [Link_CurrentAction] ... [/Link_CurrentAction] and setting -MaxRecords to 1. This method allows the visitor to continue navigating by records on the detail page.

To create a link to a particular record:

There are two format files involved in most detail links. The listing format file `default.lasso` includes the [Inline] ... [/Inline] tags that define the search for the found set. The detail format file `response.lasso` includes the [Inline] ... [/Inline] tags that find and display the individual record.

- 1 The [Inline] tag in `default.lasso` simply performs a -FindAll action. Each record in the result set is displayed with a link to `response.lasso` created using the [Link_Detail] ... [/Link_Detail] tags.

```

[Inline:-FindAll,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',

```

```

    -MaxRecords=4]
[Link_SetFormat: -NoClassic]
[Records]
  <br>[Link_Detail: -Response='response.lasso']
    [Field: 'First_Name'] [Field: 'Last_Name']
  [/Link_Detail]
[/Records]
[/Inline]
→ <br>Jane Doe
  <br>John Person
  <br>Jane Person
  <br>John Doe

```

- 2 The [Inline] tag on response.lasso uses [Action_Params] to pull the values from the URL generated by the link tags. The results contain more information about the particular records than is shown in the listing. In this case, the Phone_Number field is included as well as the First_Name and Last_Name.

```

[Inline:(Action_Params),
  -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
<br>[Field: 'First_Name'] [Field: 'Last_Name']
<br>[Field: 'Phone_Number']
...
[/Inline]
→ <br>Jane Doe
  <br>555-1212

```

To create a link to the current record in the found set:

There are two format files involved in most detail links. The listing format file default.lasso includes the [Inline] ... [/Inline] tags that define the search for the found set. The detail format file response.lasso includes the [Inline] ... [/Inline] tags that find and display the individual record. The [Link_CurrentAction] ... [/Link_CurrentAction] tags are used to create a link from default.lasso to response.lasso showing a particular record.

- 1 The [Inline] tag on default.lasso simply performs a -FindAll action. Each record in the result set is displayed with a link to response.lasso created using the [Link_CurrentAction] ... [/Link_CurrentAction] tag.

```

[Inline:-FindAll,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords=4]

```

```

[Link_SetFormat: -NoClassic]
[Records]
  <br>[Link_CurrentAction: -Response='response.lasso', -MaxRecords=1]
    [Field: 'First_Name'] [Field: 'Last_Name']
  [/Link_CurrentAction]
[/Records]
[/Inline]
→ <br>Jane Doe
   <br>John Person
   <br>Jane Person
   <br>John Doe

```

- 2 The [Inline] tag in response.lasso uses [Action_Params] to pull the values from the URL generated by the link tags. The results contain more information about the particular records than is shown in the listing. In this case, the Phone_Number field is included as well as the First_Name and Last_Name.

The detail page can also contain links to the previous and next records in the found set. These are created using the [Link_PrevRecord] ... [/Link_PrevRecord] and [Link_NextRecord] ... [/Link_NextRecord] tags. The visitor can continue navigating the found set record by record.

```

[Inline:(Action_Params),
  -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
<br>[Field: 'First_Name'] [Field: 'Last_Name']
<br>[Field: 'Phone_Number']
...
[Link_SetFormat: -NoClassic]
<br>[Link_PrevRecord] Previous Record [/Link_PrevRecord]
<br>[Link_NextRecord] Next Record [/Link_NextRecord]
[/Inline]
→ <br>Jane Last_Name
   <br>555-1212
   <br>Previous Record
   <br>Next Record

```

8

Chapter 8

Adding and Updating Records

This chapter documents the LDML command tags which add, update, delete, and duplicate records within Lasso compatible databases.

- *Overview* provides an introduction to the database actions described in this chapter and presents important security considerations.
- *Adding Records* includes requirements and instructions for adding records to a database.
- *Updating Records* includes requirements and instructions for updating records within a database.
- *Deleting Records* includes requirements and instructions for deleting records within a database.
- *Duplicating Records* includes requirements and instructions for duplicating records within a database.

Overview

LDML provides command tags for adding, updating, deleting, and duplicating records within Lasso compatible databases. These command tags are used in conjunction with additional command tags and name/value parameters in order to perform the desired database action in a specific database and table or within a specific record.

The command tags documented in this chapter are listed in *Table 1: Command Tags*. The sections that follow describe the additional command tags and name/value parameters required for each database action.

Table 1: Command Tags

Tag	Description
-Add	Adds a record to a database.
-Update	Updates a record within a database.
-Delete	Removes a record from a database.
-Duplicate	Duplicates a record within a database. Works with FileMaker Pro databases.

Character Encoding

Lasso stores and retrieves data from data sources based on the preferences established in the **Setup > Data Sources** section of Lasso Administration. The following rules apply for each standard data source.

Lasso MySQL and MySQL – By default all communication is in the Latin-1 (ISO 8859-1) character set. This is to preserve backwards compatibility with prior versions of Lasso. The character set can be changed to the Unicode standard UTF-8 character set in the **Setup > Data Sources > Tables** section of Lasso Administration.

FileMaker Pro – By default all communication is in the MacRoman character set when Lasso Professional is hosted on Mac OS X or in the Latin-1 (ISO 8859-1) character set when Lasso Professional is hosted on Windows. The preference in the **Setup > Data Sources > Databases** section of Lasso Administration can be used to change the character set for cross-platform communications.

JDBC – All communication with JDBC data sources is in the Unicode standard UTF-8 character set.

See the Lasso Professional 7 Setup Guide for more information about how to change the character set settings in Lasso Administration.

Error Reporting

After a database action has been performed, Lasso reports any errors which occurred via the `[Error_CurrentError]` tag. The value of this tag should be checked to ensure that the database action was successfully performed.

To display the current error code and message:

The following code can be used to display the current error message. This code should be placed in a format file which is a response to a database action or within a pair of `[Inline] ... [/Inline]` tags.

```
[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
```


If the database action was performed successfully then the following result will be returned.

0: No Error

To check for a specific error code and message:

The following example shows how to perform code to correct or report a specific error if one occurs. The following example uses a conditional `[If] ... [/If]` tag to check the current error message and see if it is equal to `[Error_AddError]`.

```
[If: (Error_CurrentError) == (Error_AddError)]
  An Add Error has occurred!
[/If]
```

Full documentation about error tags and error codes can be found in *Chapter 21: Error Control*. A list of all Lasso error codes and messages can be found in *Appendix B: Error Codes*.

Classic Lasso

If Classic Lasso support has been disabled within Lasso Administration then database actions will not be performed automatically if they are specified within HTML forms or URLs. Although the database action will not be performed, the `-Response` tag will function normally. Use the following code in the response page to the HTML forms or URL to trigger the database action.

```
[Inline: (Action_Params)]
[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
[/Inline]
```

See *Chapter 6: Database Interaction Fundamentals* and *Chapter 6: Setting Global Preferences* of the Lasso Professional 7 Setup Guide for more information.

Security

Lasso has a robust internal security system that can be used to restrict access to database actions or to allow only specific users to perform database actions. If a database action is attempted when the current visitor has insufficient permissions then they will be prompted for a username and password. An error will be returned if the visitor does not enter a valid username and password.

An `[Inline] ... [/Inline]` can be specified to execute with the permissions of a specific user by specifying `-Username` and `-Password` command tags within the `[Inline]` tag. This allows the database action to be performed even though

the current site visitor does not necessarily have permissions to perform the database action. In essence, a valid username and password are embedded into the format file.

Table 2: Security Command Tags

Tag	Description
-Username	Specifies the username from Lasso Security which should be used to execute the database action.
-Password	Specifies the password which corresponds to the username.

To specify a username and password in an [Inline]:

The following example shows a -Delete action performed within an [Inline] tag using the permissions granted for username SiteAdmin with password Secret.

```
[Inline: -Delete,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -KeyValue=137,
  -Username='SiteAdmin',
  -Password='Secret']

[Error_CurrentError: -ErrorCode]: [Error_CurrentError]

[/Inline]
```

A specified username and password is only valid for the [Inline] ... [/Inline] tags in which it is specified. It is not valid within any nested [Inline] ... [/Inline] tags. See *Chapter 8: Setting Up Security* of the Lasso Professional 7 Setup Guide for additional important information regarding embedding usernames and passwords into [Inline] tags.

Adding Records

Records can be added to any Lasso compatible database using the -Add command tag. The -Add command tag requires that a number of additional command tags be defined in order to perform the -Add action. The required command tags are detailed in *Table 4: -Add Action Requirements*.

Table 3: -Add Action Requirements

Tag	Description
-Add	The action which is to be performed. Required.
-Database	The database in which the record should be added. Required.
-Table	The table from the specified database in which the record should be added. Required.
-KeyField	The name of the field which holds the primary key for the specified table. Recommended.
Name/Value Parameters	A variable number of name/value parameters specify the initial field values for the added record. Optional.

Any name/value parameters included in the -Add action will be used to set the starting values for the record which is added to the database. All name/value parameters must reference a writable field within the database. Any fields which are not referenced will be set to their default values according to the database's configuration.

Lasso returns a reference to the record which was added to the database. The reference is different depending on what type of database to which the record was added.

- **Lasso MySQL and MySQL** – The -KeyField tag should be set to the primary key field or auto-increment field of the table. Lasso will return the added record as the result of the action by checking the specified key field for the last insterted record. The [KeyField_Value] tag can be used to inspect the value of the auto-increment field for the inserted record.

If no -KeyField is specified, the specified -KeyField is not an auto-increment field, or -MaxRecords is set to 0 then no record will be returned as a result of the -Add action. This can be useful in situations where a large record is being added to the database and there is no need to inspect the values which were added.

- **FileMaker Pro** – The [KeyField_Value] tag is set to the value of the internal Record ID for the new record. The Record ID functions as an auto-increment field that is automatically maintained by FileMaker Pro for all records.

FileMaker Pro automatically performs a search for the record which was added to the database. The found set resulting from an -Add action is equivalent to a search for the single record using the [KeyField_Value].

The value for -KeyField is ignored when adding records to a FileMaker Pro database. The value for [KeyField_Value] is always the internal Record ID value.

Note: Consult the documentation for third-party data sources to see what behavior they implement when adding records to the database.

To add a record using [Inline] ... [/Inline] tags:

The following example shows how to perform an -Add action by specifying the required command tags within an opening [Inline] tag. -Database is set to Contacts, -Table is set to People, and -KeyField is set to ID. Feedback that the -Add action was successful is provided to the visitor inside the [Inline] ... [/Inline] tags using the [Error_CurrentError] tag. The added record will only include default values as defined within the database itself.

```
[Inline: -Add,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID']
  <p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
[/Inline]
```

If the -Add action is successful then the following will be returned.

→ 0: No Error

To add a record with data using [Inline] ... [/Inline] tags:

The following example shows how to perform an -Add action by specifying the required command tags within an opening [Inline] tag. In addition, the [Inline] tag includes a series of name/value parameters that define the values for various fields within the record that is to be added. The First_Name field is set to John and the Last_Name field is set to Doe. The added record will include these values as well as any default values defined in the database itself.

```
[Inline: -Add,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  'First_Name'='John',
  'Last_Name'='Doe']
  <p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
  <br>Record [Field: 'ID'] was added for [Field: 'First_Name'] [Field: 'Last_Name'].
[/Inline]
```

The results of the -Add action contain the values for the record that was just added to the database.

→ 0: No Error
Record 2 was added for John Doe.

To add a record using an HTML form:

The following example shows how to perform an -Add action using an HTML form to send values into an [Inline] tag through [Action_Params]. The text inputs provide a way for the site visitor to define the initial values for various fields in the record which will be added to the database. The site visitor can set values for the fields First_Name and Last_Name.

```
<form action="response.lasso" method="POST">
  <br>First Name: <input type="text" name="First_Name" value="">
  <br>Last Name: <input type="text" name="Last_Name" value="">
  <br><input type="submit" name="-Nothing" value="Add Record">
</form>
```

The response page for the form, response.lasso, contains the following code that performs the action using an [Inline] tag and provides feedback that the record was successfully added to the database. The field values for the record that was just added to the database are automatically available within the [Inline] ... [/Inline] tags.

```
[Inline: (Action_Params),
  -Add,
  -Database='Contacts',
  -Table='People',
  -Keyfield='ID']
<p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
<br>Record [Field: 'ID'] was added for [Field: 'First_Name'] [Field: 'Last_Name'].
[/Inline]
```

If the form is submitted with Mary in the First Name input and Person in the Last Name input then the following will be returned.

```
→ 0: No Error
   Record 3 was added for Mary Person
```

To add a record using a URL:

The following example shows how to perform an -Add action using a URL to send values into an [Inline] tag through [Action_Params]. The name/value parameters in the URL define the starting values for various fields in the database: First_Name is set to John and Last_Name is set to Person.

```
<a href="response.lasso?First_Name=John&Last_Name=Person">
  Add John Person
</a>
```

The response page for the URL, response.lasso, contains the following code that performs the action using [Inline] tag and provides feedback that the record was successfully added to the database. The field values for the

record that was just added to the database are automatically available within the [Inline] ... [/Inline] tags.

```
[Inline: (Action_Params),  
  -Add,  
  -Database='Contacts',  
  -Table='People',  
  -Keyfield='ID']  
<p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]  
<br>Record [Field: 'ID'] was added for [Field: 'First_Name'] [Field: 'Last_Name'].  
[/Inline]
```

If the link for Add John Person is selected then the following will be returned.

```
→ 0: No Error  
   Record 4 was added for John Person.
```

Updating Records

Records can be updated within any Lasso compatible database using the -Update command tag. The -Update command tag requires that a number of additional command tags be defined in order to perform the -Update action. The required command tags are detailed in *Table 5: -Update Action Requirements*.

Table 4: -Update Action Requirements

Tag	Description
-Update	The action which is to be performed. Required.
-Database	The database in which the record should be added. Required.
-Table	The table from the specified database in which the record should be added. Required.
-KeyField	The name of the field which holds the primary key for the specified table. Required.
-KeyValue	The value of the primary key of the record which is to be updated. Required.
Name/Value Parameters	A variable number of name/value parameters specifying the field values which need to be updated. Optional.

Lasso identifies the record which is to be updated using the values for the command tags -KeyField and -KeyValue. -KeyField must be set to a field in the table which has a unique value for every record in the table. Usually, this is the primary key field for the table. -KeyValue must be set to a valid value

for the `-KeyField` in the table. If no record can be found with the specified `-KeyValue` then an error will be returned.

Any name/value parameters included in the update action will be used to set the field values for the record which is updated. All name/value parameters must reference a writable field within the database. Any fields which are not referenced will maintain the values they had before the update.

Lasso returns a reference to the record which was updated within the database. The reference is different depending on what type of database is being used.

- **Lasso MySQL and MySQL** – The `[KeyField_Value]` tag is set to the value of the key field which was used to identify the record to be updated. The `-KeyField` should always be set to the primary key or auto-increment field of the table. The results when using other fields are undefined.

If the `-KeyField` is not set to the primary key field or auto-increment field of the table or if `-MaxRecords` is set to 0 then no record will be returned as a result of the `-Update` action. This is useful if a large record is being updated and the results of the update do not need to be inspected.

- **FileMaker Pro** – The `[KeyField_Value]` tag is set to the value of the internal Record ID for the updated record. The Record ID functions as an auto-increment field that is automatically maintained by FileMaker Pro for all records.

Lasso automatically performs a search for the record which was updated within the database. The found set resulting from an `-Update` action is equivalent to a search for the single record using the `[KeyField_Value]`.

Note: Consult the documentation for third-party data sources to see what behavior they implement when updating records within a database.

To update a record with data using `[Inline] ... [/Inline]` tags:

The following example shows how to perform an `-Update` action by specifying the required command tags within an opening `[Inline]` tag. The record with the value 2 in field ID is updated. The `[Inline]` tag includes a series of name/value parameters that define the new values for various fields within the record that is to be updated. The `First_Name` field is set to Bob and the `Last_Name` field is set to Surname. The updated record will include these new values, but any fields which were not included in the action will be left with the values they had before the update.

```

[Inline: -Update,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -KeyValue=2,
  'First_Name'='Bob',
  'Last_Name'='Surname']

<p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
<br>Record [Field: 'ID'] was added for [Field: 'First_Name'] [Field: 'Last_Name'].

[/Inline]

```

The updated field values from the -Update action are automatically available within the [Inline].

→ 0: No Error
Record 2 was updated to Bob Surname.

To update a record using an HTML form:

The following example shows how to perform an -Update action using an HTML form to send values into an [Inline] tag. The text inputs provide a way for the site visitor to define the new values for various fields in the record which will be updated in the database. The site visitor can see and update the current values for the fields First_Name and Last_Name.

```

[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -KeyValue=3]

<form action="response.lasso" method="POST">
  <input type="hidden" name="-KeyValue" value="[KeyField_Value]">

  <br>First Name: <input type="text" name="First_Name"
    value="[Field: 'First_Name']">
  <br>Last Name: <input type="text" name="Last_Name"
    value="[Field: 'Last_Name']">
  <br><input type="submit" name="-Update" value="Update Record">
</form>

[/Inline]

```

The response page for the form, response.lasso, contains the following code that performs the action using an [Inline] tag and provides feedback that the

record was successfully updated in the database. The field values from the updated record are available automatically within the [Inline] ... [/Inline] tags.

```
[Inline: (Action_Params),
  -Update,
  -Database='Contacts',
  -Table='People',
  -Keyfield='ID']
<p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
<br>Record [Field: 'ID'] was updated to [Field: 'First_Name'] [Field: 'Last_Name'].
[/Inline]
```

The form initially shows Mary for the First Name input and Person for the Last Name input. If the form is submitted with the Last Name changed to Peoples then the following will be returned. The First Name field is unchanged since it was left set to Mary.

→ 0: No Error
Record 3 was updated to Mary Peoples.

To update a record using a URL:

The following example shows how to perform an -Update action using a URL to send field values to an [Inline] tag. The name/value parameters in the URL define the new values for various fields in the database: First_Name is set to John and Last_Name is set to Person.

```
<a href="response.lasso?-KeyValue=4&
  First_Name=John&Last_Name=Person"> Update John Person </a>
```

The response page for the URL, response.lasso, contains the following code that performs the action using [Inline] ... [/Inline] tags and provides feedback that the record was successfully updated within the database.

```
[Inline: (Action_Params),
  -Update,
  -Database='Contacts',
  -Table='People',
  -Keyfield='ID']
<p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
<br>Record [Field: 'ID'] was updated to [Field: 'First_Name'] [Field: 'Last_Name'].
[/Inline]
```

If the link for Update John Person is submitted then the following will be returned.

→ 0: No Error
Record 4 was updated for John Person.

To update several records at once:

The following example shows how to perform an -Update action on several records at once within a single database table. The goal is to update every record in the database with the last name of Person to the new last name of Peoples.

The outer [Inline] ... [/Inline] tags perform a search for all records in the database with Last_Name equal to Person. This forms the found set of records which need to be updated. The [Records] ... [/Records] tags repeat once for each record in the found set. The -MaxRecords='All' command tag ensures that all records which match the criteria are returned.

The inner [Inline] ... [/Inline] tags perform an update on each record in the found set. Substitution tags are used to retrieve the values for the required command tags -Database, -Table, -KeyField, and -KeyValue. This ensures that these values match those from the outer [Inline] ... [/Inline] tags exactly. The name/value pair 'Last_Name='Peoples' updates the field to the new value.

```
[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords='All',
  'Last_Name='Person']
[Records]
  [Inline: -Update,
    -Database=(Database_Name),
    -Table=(Table_Name),
    -KeyField=(KeyField_Name),
    -KeyValue=(KeyField_Value),
    'Last_Name='Peoples']

    <p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
    <br>Record [Field: 'ID'] was updated to
    [Field: 'First_Name'] [Field: 'Last_Name'].

  [/Inline]
[/Records]
[/Inline]
```

This particular search only finds one record to update. If the update action is successful then the following will be returned for each updated record.

➔ 0: No Error
Record 4 was updated to John Peoples.

Deleting Records

Records can be deleted from any Lasso compatible database using the `-Delete` command tag. The `-Delete` command tag can be specified within an `[Inline]` tag, an HTML form, or a URL. The `-Delete` command tag requires that a number of additional command tags be defined in order to perform the `-Delete` action. The required command tags are detailed in *Table 6: -Delete Action Requirements*.

Table 5: -Delete Action Requirements

Tag	Description
<code>-Delete</code>	The action which is to be performed. Required.
<code>-Database</code>	The database in which the record should be added. Required.
<code>-Table</code>	The table from the specified database in which the record should be added. Required.
<code>-KeyField</code>	The name of the field which holds the primary key for the specified table. Required.
<code>-KeyValue</code>	The value of the primary key of the record which is to be deleted. Required.

Lasso identifies the record which is to be deleted using the values for the command tags `-KeyField` and `-KeyValue`. `-KeyField` must be set to a field in the table which has a unique value for every record in the table. Usually, this is the primary key field for the table. `-KeyValue` must be set to a valid value for the `-KeyField` in the table. If no record can be found with the specified `-KeyValue` then an error will be returned.

Lasso returns an empty found set in response to a `-Delete` action. Since the record has been deleted from the database the `[Field]` tag can no longer be used to retrieve any values from it. The `[Error_CurrentError]` tag should be checked to ensure that it has a value of `No Error` in order to confirm that the record has been successfully deleted.

There is no confirmation or undo of a delete action. When a record is removed from a database it is removed permanently. It is important to set up Lasso security appropriately so accidental or unauthorized deletes don't occur. See *Chapter 8: Setting Up Security* in the Lasso Professional 7 Setup Guide for more information about setting up database security.

To delete a record with data using `[Inline]` ... `/[Inline]` tags:

The following example shows how to perform a delete action by specifying the required command tags within an opening `[Inline]` tag. The record with the value 2 in field ID is deleted.

```

[Inline: -Delete,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -KeyValue=2]

<p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]

[/Inline]

```

If the delete action is successful then the following will be returned.

→ 0: No Error

To delete several records at once:

The following example shows how to perform a -Delete action on several records at once within a single database table. The goal is to delete every record in the database with the last name of Peoples.

Warning: This technique can be used to remove all records from a database table. It should be used with extreme caution and tested thoroughly before being added to a public Web site.

The outer [Inline] ... [/Inline] tags perform a search for all records in the database with Last_Name equal to Peoples. This forms the found set of records which need to be updated. The [Records] ... [/Records] tags repeat once for each record in the found set. The -MaxRecords='All' command tag ensures that all records which match the criteria are returned.

The inner [Inline] ... [/Inline] tags delete each record in the found set. Substitution tags are used to retrieve the values for the required command tags -Database, -Table, -KeyField, and -KeyValue. This ensures that these values match those from the outer [Inline] ... [/Inline] tags exactly.

```

[Inline: -Search,
  -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -MaxRecords='All',
  'Last_Name'='Peoples']
[Records]

  [Inline: -Delete,
    -Database=(Database_Name),
    -Table=(Table_Name),
    -KeyField=(KeyField_Name),
    -KeyValue=(KeyField_Value)]

    <p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]

  [/Inline]

```

```
[/Records]
[/Inline]
```

This particular search only finds one record to delete. If the delete action is successful then the following will be returned for each deleted record.

→ 0: No Error

Duplicating Records

Records can be duplicated within any Lasso compatible database using the `-Duplicate` command tag. The `-Duplicate` command tag can be specified within an `[Inline]` tag, an HTML form, or a URL. The `-Duplicate` command tag requires that a number of additional command tags be defined in order to perform the `-Duplicate` action. The required command tags are detailed in *Table 7: -Duplicate Action Requirements*.

Note: Lasso Connector for Lasso MySQL and Lasso Connector for MySQL do not support the `-Duplicate` command tag.

Table 6: -Duplicate Action Requirements

Tag	Description
<code>-Duplicate</code>	The action which is to be performed. Required.
<code>-Database</code>	The database in which the record should be added. Required.
<code>-Table</code>	The table from the specified database in which the record should be added. Required.
<code>-KeyField</code>	The name of the field which holds the primary key for the specified table. Required.
<code>-KeyValue</code>	The value of the primary key of the record which is to be duplicated. Required.
Name/Value Parameters	A variable number of name/value parameters specifying field values which should be modified in the duplicated record. Optional.

Lasso identifies the record which is to be duplicated using the values for the command tags `-KeyField` and `-KeyValue`. `-KeyField` must be set to a field in the table which has a unique value for every record in the table. Usually, this is the primary key field for the table. `-KeyValue` must be set to a valid value for the `-KeyField` in the table. If no record can be found with the specified `-KeyValue` then an error will be returned.

Any name/value parameters included in the duplicate action will be used to set the field values for the record which is added to the database. All

name/value parameters must reference a writable field within the database. Any fields which are not referenced will maintain the values they had from the record which was duplicated.

Lasso always returns a reference to the new record which was added to the database as a result of the -Duplicate action. This is equivalent to performing a -Search action which returns a single record found set containing just the record which was added to the database.

To duplicate a record with data using [Inline] ... [/Inline] tags:

The following example shows how to perform a duplicate action within a FileMaker Pro database by specifying the required command tags within an opening [Inline] tag. The record with the value 2 for the keyfield value is duplicated. The [Inline] tag includes a series of name/value parameters that define the new values for various fields within the record that is to be updated. The First_Name field is set to Joe and the Last_Name field is set to Surname. The new record will include these values, but any fields which were not specified in the action will be left with the values they had from the source record.

```
[Inline: -Duplicate,
  -Database='Contacts.fp3',
  -Table='People',
  -KeyField='ID',
  -KeyValue=2,
  'First_Name'='Joe',
  'Last_Name'='Surname']
  <p>[Error_CurrentError: -ErrorCode]: [Error_CurrentError]
  <br>Record [Field: 'ID'] was duplicated for [Field: 'First_Name'] [Field: 'Last_Name'].
[/Inline]
```

If the duplicate action is successful then the following will be returned. The values from the [Field] tags are retrieved from the record which was just added to the database as a result of the duplicate action.

➔ 0: No Error
Record 6 was duplicated for Joe Surname.

9

Chapter 9

MySQL Data Sources

This chapter documents tags and behaviors which are specific to MySQL data sources, including the built-in Lasso MySQL data source.

- *Overview* introduces MySQL data sources.
- *MySQL Tags* describes tags specific to MySQL data sources.
- *Searching Records* describes unique search operations that can be performed using MySQL data sources.
- *Adding and Updating Records* describes unique add and update operations that can be performed using MySQL data sources.
- *Value Lists* describes how to retrieve and show lists of allowed field values for ENUM and SET fields in MySQL data sources.
- *Creating Database Tables* describes the [Database_...] tags that can be used to create, change, or remove tables and fields within MySQL data sources.

Overview

Lasso Professional 7 includes a built-in Lasso MySQL data source and allows for a connection to a remote MySQL data source to be established. This chapter primarily documents tags and features unique to Lasso MySQL or other MySQL data sources.

Since Lasso MySQL is the Lasso Professional 7 default data source, all of the documentation and examples in this manual are targeted for Lasso MySQL except when explicitly stated otherwise. All of the procedures outlined in *Chapter 6: Database Interaction Fundamentals*, *Chapter 7: Searching and Displaying Data*, and *Chapter 8: Adding and Updating Records* can be used with Lasso MySQL.

Note: The tags and procedures defined in this chapter are primarily for use with MySQL data sources including Lasso MySQL. Many of the tags and procedures will work with any SQL-based data source with minor variations, if any.

Tips for Using MySQL Data Sources

- Always specify a primary key field using the `-KeyField` command tag in `-Search`, `-Add`, and `-Findall` actions. This will ensure that the `[KeyField_Value]` tag will always return a value.
- Use `-KeyField` and `-KeyValue` to reference a particular record for updates, duplicates, or deletes.
- MySQL data sources are case-sensitive. For best results, reference MySQL database and table names in the same letter-case as they appear on disk in your LDML code.
- MySQL fields truncate any data beyond the length they are set up to store. Ensure that all fields in MySQL databases have sufficiently long fields for the values that need to be stored in them.
- Use `-ReturnField` command tags to reduce the number of fields which are returned from a `-Search` action. Returning only the fields that need to be used for further processing or shown to the site visitor reduces the amount of data that needs to travel between Lasso Service and Lasso MySQL.
- When an `-Add` or `-Update` action is performed on a MySQL database, the data from the added or updated record is returned inside the `[Inline] ... [Inline]` tags or alternately to the Classic Lasso response page. If the `-ReturnField` parameter is used, then only those fields specified should be returned from an `-Add` or `-Update` action. Setting `-MaxRecords=0` can be used as an indication that no record should be returned.
- See *Chapter 9: Administration Utilities* in the Lasso Professional 7 Setup Guide for information about optimizing tables for optimum performance and checking tables for damage.

Security Tips

- The `-SQL` command tag can only be allowed or disallowed at the host level for users in Lasso Administration. Once the `-SQL` command tag is allowed for a user, that user may access any database within the allowed host inside of a SQL statement. For that reason, only trusted users should be allowed to issue SQL queries using the `-SQL` command tag. For more information, see *Chapter 8: Setting Up Security* in the Lasso Professional 7 Setup Guide.

- SQL statements which are generated using visitor-defined data should be screened carefully for unwanted commands such as DROP or GRANT. See *Chapter 7: Setting Up Data Sources* of the Lasso Professional 7 Setup Guide for more information.
- Always quote any inputs from site visitors that are incorporated into SQL statements. For example, the following SQL SELECT statement includes quotes around the [Action_Param] value. The quotes are escaped \' so they will be embedded within the string rather than ending the string literal. The semi-colon at the end of the statement is optional unless multiple statements are issued.

```
[Variable: 'SQL_Statement'='SELECT * FROM Contacts.People WHERE ' +  
  'First_Name LIKE \'' + (Action_Param: 'First_Name') + '\';']
```

If [Action_Param] returns John for First_Name then the SQL statement generated by this code would appear as follows.

```
SELECT * FROM Contacts.People WHERE First_Name LIKE 'John';
```

MySQL Tags

LDML 7 includes tags to identify which type of MySQL data source is being used. These tags are summarized in *Table 1: Enhanced MySQL Tags*.

Table 1: Enhanced MySQL Tags

Tag	Description
[Lasso_DatasourcelsLassoMySQL]	Returns True if a database is hosted by Lasso MySQL. Requires one string value which is the name of a database.
[Lasso_DatasourcelsMySQL]	Returns True if a database is hosted by MySQL. Requires one string value which is the name of a database.

To check whether a database is hosted by Lasso MySQL:

The following example shows how to use [Lasso_DatasourcelsLassoMySQL] to check whether the database Example is hosted by Lasso MySQL or not.

```
[If: (Lasso_DatasourcelsLassoMySQL: 'Example')]  
  Example is hosted by Lasso MySQL!  
[Else]  
  Example is not hosted by Lasso MySQL.  
[/If]
```

➔ Example is hosted by Lasso MySQL!

To list all databases hosted by Lasso MySQL:

Use the [Database_Names] ... [/Database_Names] tags to list all databases available to Lasso. The [Lasso_DatasourcesLassoMySQL] tag can be used to check each database and only those that are hosted by Lasso MySQL will be returned. The result shows two databases, Site and Example, which are available through Lasso MySQL.

```
[Database_Names]
  [If: (Lasso_DatasourcesLassoMySQL:(Database_NameItem))]
    <br>[Database_NameItem]
  [/If]
[/Database_Names]
```

```
→ <br>Example
   <br>Site
```

Searching Records

In LDML 7, there are unique search operations that can be performed using MySQL data sources. These search operations take advantage of special functions in MySQL such as full-text indexing, regular expressions, record limits, and distinct values to allow optimal performance and power when searching. These search operations can be used on MySQL data sources in addition to all search operations described in *Chapter 7: Searching and Displaying Data*.

Search Field Operators

Additional field operators are available for the -Operator (or -Op) tag when searching MySQL data sources. These operators are summarized in *Table 2: MySQL Search Field Operators*. Basic use of the -Operator tag is described in *Chapter 7: Searching and Displaying Data*.

Table 2: MySQL Search Field Operators

Operator	Description
ft	Full-Text Search. If used, a MySQL full-text search is performed on the field specified. Will only work on fields that are full-text indexed in MySQL. Records are automatically returned in order of high relevance (contains many instances of that value) to low relevance (contains few instances of the value). Only one ft operator may be used per action, and no -SortField parameter should be specified.

rx	Regular Expression. If used, then regular expressions may be used as part of the search field value. Returns records matching the regular expression value for that field.
nrx	Not Regular Expression. If used, then regular expressions may be used as part of the search field value. Returns records that do not match the regular expression value for that field.

Note: For more information on full-text searches and regular expressions supported in MySQL, see the MySQL documentation.

To perform a full-text search on a field:

If a MySQL field is indexed as full-text, then using -Op='ft' before the field in a search inline performs a MySQL full text search on that field. The example below performs a full text search on the Jobs field in the Contacts database, and returns the First_Name field for each record that contain the word Manager. Records that contain the most instances of the word Manager are returned first.

```
[Inline: -Search, -Database='Contacts', -Table='People',
-Op='ft',
'Jobs'='Manager']
[Records]
[Field:'First_Name']<br>
[/Records]
[/Inline]
```

→ Mike

Jane

To use regular expressions as part of a search:

Regular expressions can be used as part of a search value for a field by using -Op='rx' before the field in a search inline. The following example searches for all records where the Last_Name field contains eight characters using a regular expression.

```
[Inline: -Search, -Database='Contacts', -Table='People',
-Op='rx',
'Last_Name'='.{8}',
-MaxRecords='All']
[Records]
[Field:'Last_Name'], [Field:'First_Name']<br>
[/Records]
[/Inline]
```

→ Lastname, Mike

 Lastname, Mary Beth

The following example searches for all records where the Last_Name field doesn't contain eight characters. This is easily accomplished using the same inline search above using -Op='nrx' instead.

```
[Inline: -Search, -Database='Contacts', -Table='People',  
-Op='nrx',  
'Last_Name'='. {8}',  
-MaxRecords='All']  
[Records]  
  [Field:'Last_Name'], [Field:'First_Name']<br>  
[/Records]  
[/Inline]
```

→ Doe, John

 Doe, Jane

 Surname, Bob

 Surname, Jane

 Surname, Margaret

 Unknown, Thomas

Search Command Tags

Additional search command tags are available when searching MySQL data sources using the [Inline] tag. These tags allow special search functions specific to MySQL to be performed without writing SQL statements. These operators are summarized in *Table 3: MySQL Search Command Tags*.

Table 3: MySQL Search Command Tags

Tag	Description
-UseLimit	Prematurely ends a -Search or FindAll action once the specified number of records for the -MaxRecords tag have been found and returns the found records. Requires the -MaxRecords tag. This issues an internal LIMIT statement to MySQL to cause it to search more efficiently.
-SortRandom	Sorts returned records randomly. Is used in place of the -SortField and -SortOrder parameters. Does not require a value.
-Distinct	Causes a -Search action to only output records that contain unique field values (comparing only returned fields). Does not require a value. May be used with the -ReturnField parameter to limit the fields checked for distinct values.

To have MySQL immediately return records once a limit is reached:

Use the `-UseLimit` tag in the search inline. Normally, Lasso will find all records that match the inline search criteria and then pair down the results based on `-MaxRecords` and `-SkipRecords` values. The `-UseLimit` tag instructs MySQL to terminate the specified search process once the number of records specified for `-MaxRecords` is found. The following example searches the `Contacts` database with a limit of five records.

```
[Inline: -FindAll,
-Database='Contacts', -Table='People',
-MaxRecords='5',
-UseLimit]
[Found_Count]
[/Inline]
```

→ 5

Note: If the `-UseLimit` tag is used, the value of the `[Found_Count]` tag will always be the same as the `-MaxRecords` value if the limit is reached. Otherwise, the `[Found_Count]` tag will return the total number of records in the specified table that match the search criteria if `-UseLimit` is not used.

To sort results randomly:

Use the `-SortRandom` tag in a search inline. The following example finds all records and sorts first by last name then randomly.

```
[Inline: -FindAll, -Database='Contacts', -Table='People',
-Keyfield='ID',
-SortRandom]
[Records]
[Field:'ID']
[/Records]
[/Inline]
```

→ 5 2 8 1 3 6 4 7

Note: Due to the nature of the `-SortRandom` tag, the results of this example will vary upon each execution of the inline.

To return only unique records in a search:

Use the `-Distinct` parameter in a search inline. The following example only returns records that contain distinct values for the `Last_Name` field.

```

[Inline: -FindAll, -Database='Contacts', -Table='People',
-ReturnField='Last_Name',
-Distinct]
[Records]
[Field:'Last_Name']<br>
[/Records]
[/Inline]
→ Doe<br>
Surname<br>
Lastname<br>
Unknown<br>

```

The `-Distinct` tag is especially useful for generating lists of values that can be used in a pull-down menu. The following example is a pull-down menu of all the last names in the `Contacts` database.

```

[Inline: -Findall, -Database='Contacts', -Table='People',
-ReturnField='Last_Name',
-Distinct]
<select name="Last_Name">
[Records]
<option value="[Field: 'Last_Name']">
[Field: 'Last_Name']
</Option>
[/Records]
</Select>
[/Inline]

```

Searching Null Values

When searching MySQL tables, NULL values may be explicitly searched for within fields using the `[Null]` tag. A NULL value in MySQL designates that there is no other value stored in that particular field. This is similar to searching a field for an empty string (e.g. `'fieldname'=""`), however NULL values and empty strings are not the same in MySQL. For more information about NULL values, see the MySQL documentation.

```

[Inline: -Search,
-Database='Contacts', -Table='People',
-Op='eq',
'Title'=(Null),
-MaxRecords='All']
[Records]
Record [Field:'ID'] does not have a title.<br>
[/Records]
[/Inline]

```

→ Record 7 does not have a title.

Record 8 does not have a title.

Adding and Updating Records

In LDML 7, there are special add and update operations that can be performed using MySQL data sources in addition to all add and update operations described in *Chapter 8: Adding and Updating Data*.

Multiple Field Values

When adding or updating data to a field in MySQL, the same field name can be used several times in an -Add or -Update inline. The result is that all data added or updated in each instance of the field name will be concatenated in a comma-delimited form. This is particularly useful for adding data to SET field types.

To add or update multiple values to a field:

The following example adds a record with two comma delimited values in the Jobs field:

```
[Inline: -Add, -Database='Contacts', -Table='People',  
-KeyField='ID',  
'Jobs'='Customer Service',  
'Jobs'='Sales']  
[Field:'Title']  
[/Inline]
```

→ Customer Service, Sales

The following example updates the Jobs field of a record with three comma-delimited values:

```
[Inline: -Update, -Database='Contacts', -Table='People',  
-KeyField='ID',  
-KeyValue='5',  
'Jobs'='Customer Service',  
'Jobs'='Sales',  
'Jobs'='Support']  
[Field:'Title']  
[/Inline]
```

→ Customer Service, Sales, Support

Note: The individual values being added or updated should not contain commas.

Adding or Updating Null Values

NULL values can be explicitly added to MySQL fields using the [Null] tag. A NULL value in MySQL designates that there is no value for a particular field. This is similar to setting a field to an empty string (e.g. 'fieldname="'), however the two are different in MySQL. For more information about NULL values, see the MySQL documentation.

To add or update a null value to a field:

Use the [Null] tag as the field value. The following example adds a record with a NULL value in the Last_Name field.

```
[Inline: -Add, -Database='Contacts', -Table='People',
-KeyField='ID',
'Last_Name'=(Null)]
[/Inline]
```

The following example updates a record with a NULL value in the Last_Name field.

```
[Inline: -Update, -Database='Contacts', -Table='People',
-KeyField='ID',
-KeyValue='5',
'Last_Name'=(Null)]
[/Inline]
```

Value Lists

A value list in Lasso is a set of possible values that can be used for a field. Value lists in MySQL are lists of pre-defined and stored values for a SET or ENUM field type. A value list from a SET or ENUM field can be displayed using the tags defined in *Table 4: MySQL Value List Tags*.

Table 4: MySQL Value List Tags

Tag	Description
[Value_List] ... [/Value_List]	Container tag repeats each value allowed for ENUM or SET fields. Requires a single parameter: the name of an ENUM or SET field from the current table.
[Value_ListItem]	Returns the value for the current item in a value list. Optional -Checked or -Selected parameter returns only values currently contained in the ENUM or SET field.
[Selected]	Displays the word Selected if the current value list item is contained in the data of the ENUM or SET field.
[Checked]	Displays the word Checked if the current value list item is contained in the data of the ENUM or SET field.
[Option]	Generates a series of <option> tags for the value list. Requires a single parameter: the name of an ENUM or SET field from the current table.

Note: See *Chapter 7: Searching and Displaying Data* for information about the -Show command tag which is used throughout this section.

To display values for an ENUM or SET field:

- Perform a -Show action to return the schema of a MySQL database and use the [Value_List] tag to display the allowed values for an ENUM or SET field. The following example shows how to display all values from the ENUM field Title in the Contacts database. SET field value lists function in the same manner as ENUM value lists, and all examples in this section may be used with either ENUM or SET field types.


```
[Inline: -Show, -Database='Contacts', -Table='People']
  [Value_List: 'Title']
    <br>[Value_ListItem]
  [/Value_List]
[/Inline]
→ <br>Mr.
   <br>Mrs.
   <br>Ms.
   <br>Dr.
```
- The following example shows how to display all values from a value list using a named inline. The same name Values is referenced by -InlineName in both the [Inline] tag and [Value_List] tag.

```

[Inline: -InlineName='Values', -Show, -Database='Contacts', -Table='People']
[/Inline]
...
[Value_List: 'Title', -InlineName='Values']
  <br>[Value_ListItem]
[/Value_List]
→ <br>Mr.
   <br>Mrs.
   <br>Ms.
   <br>Dr.

```

To display an HTML pop-up menu in an -Add form with all values from a value list:

- The following example shows how to format an HTML `<select> ... </select>` pop-up menu to show all the values from a value list. A select list can be created with the same code by including size and/or multiple parameters within the `<select>` tag. This code is usually used within an HTML form that performs an -Add action so the visitor can select a value from the value list for the record they create.

The example shows a single `<select> ... </select>` within `[Inline] ... [/Inline]` tags with a -Show command. If many value lists from the same database are being formatted, they can all be contained within a single set of `[Inline] ... [/Inline]` tags.

```

<form action="response.lasso" method="POST">
  <input type="hidden" name="-Add" value="">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">

  [Inline: -Show, -Database='Contacts', -Table='People']
    <select name="Title">
      [Value_List: 'Title']
        <option value="[Value_ListItem]">[Value_ListItem]</option>
      [/Value_List]
    </select>
  [/Inline]

  <p><input type="submit" name="-Add" value="Add Record">
</form>

```

- The `[Option]` tag can be used to easily format a value list as an HTML `<select> ... </select>` pop-up menu. The `[Option]` tag generates all of the `<option> ... </option>` tags for the pop-up menu based on the value list for the specified field. The example below generates exactly the same HTML as the example above.

```

<form action="response.lasso" method="POST">
  <input type="hidden" name="-Add" value="">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">

  [Inline: -Show, -Database='Contacts', -Table='People']
  <select name="Title">
    [Option: 'Title']
  </select>
[/Inline]

<p><input type="submit" name="-Add" value="Add Record">
</form>

```

To display HTML radio buttons with all values from a value list:

The following example shows how to format a set of HTML `<input>` tags to show all the values from a value list as radio buttons. The visitor will be able to select one value from the value list. Check boxes can be created with the same code by changing the type from radio to checkbox.

```

<form action="response.lasso" method="POST">
  <input type="hidden" name="-Add" value="">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">

  [Inline: -Show, -Database='Contacts', -Table='People']
  [Value_List: 'Title']
  <input type="radio" name="Title" value="[Value_ListItem]" [Value_ListItem]
[/Value_List]
[/Inline]

<p><input type="submit" name="-Add" value="Add Record">
</form>

```

To display only selected values from a value list:

The following examples show how to display the selected values from a value list for the current record. The record for John Doe is found within the database and the selected value for the Title field, Mr. is displayed.

- The `-Selected` keyword in the `[Value_ListItem]` tag ensures that only selected value list items are shown. The following example uses a conditional to check whether `[Value_ListItem: -Selected]` is empty.

```

[Inline: -Search, -Database='Contacts', -Table='People',
-KeyField='ID',
-KeyValue=126]
[Value_List: 'Title']

```

```

[If: (Value_ListItem: -Selected) != ' ' ]
  <br>[Value_ListItem: -Selected]
[/If]
[/Value_List]
[/Inline]

```

→
Mr.

- The [Selected] tag ensures that only selected value list items are shown. The following example uses a conditional to check whether [Selected] is empty and only shows the [Value_ListItem] if it is not.

```

[Inline: -Search, -Database='Contacts', -Table='People',
-KeyField='ID',
-KeyValue=126]
[Value_List: 'Title']
  [If: (Selected) != ' ' ]
    <br>[Value_ListItem]
  [/If]
[/Value_List]
[/Inline]

```

→
Mr.

- The [Field] tag can also be used simply to display the current value for a field without reference to the value list.

```
<br>[Field: 'Title']
```

→
Mr.

To display an HTML pop-up menu in an -Update form with selected value list values:

- The following example shows how to format an HTML <select> ... </select> select list to show all the values from a value list with the selected values highlighted. The [Selected] tag returns Selected if the current value list item is selected in the database or nothing otherwise. This code will usually be used in an HTML form that performs an -Update action to allow the visitor to see what values are selected in the database currently and make different choices for the updated record.

```

<form action="response.lasso" method="POST">
  <input type="hidden" name="-Update" value="">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-KeyValue" value="127">

```

```
[Inline: -Search, -Database='Contacts', -Table='People',
-KeyField='ID',
-KeyValue=126]
<select name="Title" multiple size="4">
  [Value_List: 'Title']
  <option value="[Value_ListItem]" [Selected]>[Value_ListItem]</option>
[/Value_List]
</select>
[/Inline]

<p><input type="submit" name="-Update" value="Update Record">
</form>
```

- The [Option] tag automatically inserts Selected parameters as needed to ensure that the proper options are selected in the HTML select list. The example below generates exactly the same HTML as the example above.

```
<form action="response.lasso" method="POST">
  <input type="hidden" name="-Update" value="">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-KeyValue" value="127">

[Inline: -Search, -Database='Contacts', -Table='People',
-KeyField='ID',
-KeyValue=126]
<select name="Title" multiple size="4">
  [Option: 'Title']
</select>
[/Inline]

<p><input type="submit" name="-Update" value="Update Record">
</form>
```

To display HTML check boxes with selected value list values:

The following example shows how to format a set of HTML <input> tags to show all the values from a value list as check boxes with the selected check boxes checked. The [Checked] tag returns Checked if the current value list item is selected in the database or nothing otherwise. Radio buttons can be created with the same code by changing the type from checkbox to radio.

```
<form action="response.lasso" method="POST">
  <input type="hidden" name="-Update" value="">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-KeyValue" value="127">
```

```
[Inline: -Search, -Database='Contacts', -Table='People',
-KeyField='ID',
-KeyValue=126]
[Value_List: 'Title']
  <input type="checkbox" name="Title" value="[Value_ListItem]" [Checked]>
  [Value_ListItem]
[/Value_List]
[/Inline]

<p><input type="submit" name="-Update" value="Update Record">
</form>
```

Note: Storing multiple values is only supported using SET field types.

Creating Database Tables

LDML 7 includes a set of tags which allow tables and fields to be created, altered, or deleted within MySQL data sources including Lasso Professional 7's internal Lasso MySQL data source.

- A solution can create its required tables automatically the first time it is accessed.
- Temporary tables can be created which store data that is eliminated the next time Lasso MySQL is restarted.
- Interactive tools can be built which allow clients to create their own tables and populate them with data.

For a visual interface that allows Lasso MySQL databases (in addition to tables and fields) to be created and altered, see the Database Builder LassoApp. This interactive tool allows databases, tables, and fields to be created, altered, or deleted. See *Chapter 10: Building and Browsing Databases* in the Lasso Professional 7 Setup Guide for details.

Lasso stores security information about all tables and fields in an internal cache. This table must be updated whenever a new table is created, new fields within a table are added, or fields are modified. The security cache can be updated manually using the Refresh button in the *Setup > Data Sources* sections of Lasso Administration, or programatically using the [DataSource_Reload] tag. Lasso will automatically refresh the security cache whenever an unknown table or field name is used. Perform an [Inline] ... [/Inline] database action that references any new or changed tables and fields to force Lasso to update its security cache.

Warning: These tags can be used to delete entire data tables from MySQL data sources. These tags should be used with care to ensure that essential

data is not lost. If a table or field is removed there is no way to access the data that was stored in the table or field without resorting to a backup.

Table 5: Database Creation Tags

Tag	Description
[Database_CreateTable]	Creates a table. Requires a -Database parameter which specifies a MySQL database and a -Table parameter which specifies the name of the table to be created.
[Database_CreateField]	Creates a field in a table. Requires -Database and -Table parameters which specify where the field should be created and -Field and -Type parameters which give the name of the field to be created and its type. [Database_CreateColumn] is a synonym.
[Database_ChangeField]	Changes a field definition. Requires the same parameters as [Database_CreateField] in addition to an -Original parameter that specifies the field to be altered. [Database_ChangeColumn] is a synonym.
[Database_RemoveTable]	Removes a table from a database. All data in the table will be lost. Requires -Database and -Table parameters.
[Database_RemoveField]	Removes a field from a table. All data in the field will be lost. Requires -Database, -Table, and -Field parameters. [Database_RemoveColumn] is a synonym.

Tables

Tables can be created or removed from Lasso MySQL or other MySQL data sources. The following important points should be kept in mind when creating or deleting new tables.

- Table names are case sensitive in MySQL, but case insensitive in Lasso. For best results use a consistent naming convention and never rely on case to differentiate between two tables.
- Table names should start with a letter and contain only letters, numbers, and the underscore character `_`. They should not contain any spaces, periods, or other punctuation.
- New tables must be enabled within Lasso Administration before they can be accessed through Lasso.
- All tables are created with a single field automatically named `ID` that is set to be the primary key field and to auto-increment from 0. Usually, this field should be used as the primary key field for a table unless another structure is required.

- In terms of data storage, tables are equivalent to FileMaker Pro database files, not to FileMaker Pro layouts. The equivalent of many FileMaker Pro databases can be stored in a single MySQL database.

However, within Lasso security and Lasso database actions, FileMaker Pro databases and MySQL databases are treated as equivalent. FileMaker Pro layouts are treated as equivalent to MySQL tables. This makes the security model cleaner and allows for easier transition between data sources.

- When a table is removed its data is lost forever. There is no undo. See *Chapter 9: Administration Utilities* in the Lasso Professional 7 Setup Guide for information about backing up tables and data recovery.

Table 6: [Database_CreateTable] Parameters:

Parameter	Description
-Database	The name of the database in which to create the table.
-Table	The name of the table to be created. Must be unique within the database. Should start with a letter and contain only letters, numbers, or underscores.
-Temporary	Creates a temporary table that will be deleted when MySQL restarts.

To create a table:

Use the [Database_CreateTable] tag to create a new table in the specified database. The [Database_CreateTable] tag will not overwrite an existing table. The name of the new table must be unique. The following example creates a new table named Phone_Book in the database Example.

```
[Database_CreateTable: -Database='Example', -Table='Phone_Book']
```

The table initially contains one field named ID that is set to be the primary key field and to auto increment. Use the tags described in the *Fields* section below to add more fields to the new table.

Note: New tables are initially disabled in Lasso Administration. Use the *Setup > Data Sources* tab in Lasso Administration to enable new tables. In addition, the Extending Lasso Guide includes the complete source code for Admin.LassoApp which demonstrates how to enable new tables automatically.

To create a temporary table:

Use the [Database_CreateTable] tag with the -Temporary keyword to create a temporary table in the specified database. The temporary table will be deleted when MySQL restarts. The following example creates a new table named Cache in the database Example. This tag could be used in a format

file within LassoStartup to create a table that started empty each time the server hosting Lasso was restarted.

```
[Database_CreateTable: -Database='Example', -Table='Cache', -Temporary]
```

The table initially contains one field named ID that is set to be the primary key field and to auto increment. Use the tags described in the *Fields* section below to add more fields to the new table.

To remove a table:

Use the [Database_RemoveTable] tag to drop the specified table from its database. This will eliminate all data stored in the table. The following example will remove the table named Cache from the Example database.

```
[Database_RemoveTable: -Database='Example', -Table='Cache']
```

Fields

Each table created by the [Database_CreateTable] command starts with only a single ID field which Lasso creates automatically. Additional fields can be created, changed, or removed from any table in a MySQL or Lasso MySQL database. The following important points should be kept in mind when creating, changing, or removing fields.

- Field names should start with a letter and contain only letters, numbers, and the underscore character (_). They should not contain any spaces, periods, or other punctuation. See the MySQL documentation for a list of reserved names that cannot be used as field names.
- Tables can only contain a single primary key field and a single auto-increment field. Since the ID field is automatically created with both of these attributes it must be removed if a different field needs to be created as the primary key field.
- Fields should be created with the smallest data type which can hold all possible values. See the MySQL documentation at <http://www.mysql.com> for information on MySQL data types.
- When a field is removed its data is lost forever. There is no undo. See *Chapter 9: Administration Utilities* in the Lasso Professional 7 Setup Guide for information about backing up tables and data recovery.
- After many fields have been added, changed, or removed from a table it is good practice to optimize the table following the instructions in the *Optimizing Tables* section below.

Table 7: [Database_CreateField] and [Database_ChangeField] Parameters:

Parameter	Description
-Database	The name of the database in which to create the table.
-Table	The name of the table in which to create the field.
-Original	Used only with [Database_ChangeField]. The name of the original field which should be changed.
-Field	The name of the field to be created. Must be unique within the table. Should start with a letter and contain only letters, numbers, or underscores.
-Type	The type of the field. See Table 5: MySQL Field Types for a summary of possible types.
-Default	Optional default value for the field. The field will be set to this value when a new record is created that does not set this field explicitly.
-AutoIncrement	Sets a field to auto increment. Only one field in each table can be set to auto increment. The field will be set to 1 greater than the maximum value of the field each time a new record is created that does not set this field explicitly. Optional
-Key	Sets a field as the primary key field. Only one field in each table can be set to be the primary key field. Optional.
-Null	Specifies that a field can contain Null values. The default.
-NotNull	Specifies that a field cannot contain Null values. Should be set for primary key or auto-increment fields. Optional.
-AfterField	Specifies where in the table the field should be created. The new field will be inserted after the named field. Optional.
-BeforeField	Specifies that a field should be created before all other fields in a table. Optional.

The -Type parameter for [Database_CreateField] and [Database_ChangeField] can accept any of the values in *Table 5: MySQL Field Types*.

Table 8: MySQL Field Types

Data Type	Description
TINYINT	Integer less than about one hundred. 8-bit.
SMALLINT	Integer less than about 30 thousand. 16-bit.
MEDIUMINT	Integer less than 8 million. 24-bit.
INT	Integer less than 2 billion. 32-bit. Recommended.
BIGINT	Very large integer. Same range as Lasso integer data type. 64-bit.
FLOAT	Short decimal value. 32-bit.
DOUBLE	Long decimal value. Same range as Lasso decimal data type. 64-bit. Recommended.
DECIMAL(length, precision)	Fixed precision decimal value. Ranges vary depending on parameters.
CHAR(length)	Fixed length string of the specified length. Length can be from 0 to 255.
VARCHAR(length)	Variable length string of the specified length. Length can be from 1 to 255.
TEXT, BLOB	Text or binary data up to about 64 KB in length.
TINYTEXT, TINYBLOB	Text or binary data up to 255 bytes. Rarely used.
MEDIUMTEXT, MEDIUMBLOB	Text or binary data up to about 16 MB. Rarely used.
LONGTEXT, LONGBLOB	Text or binary data up to about 4 GB. Practical limit of about 24 MB. Rarely used.
ENUM ('Value1', 'Value2', ...)	A field that can contain one of a number of predefined string values that are indexed numerically. ENUM data can be text referring to a value, or an integer referring to the index number of a value. A maximum of 65,535 ENUM values may be predefined.
SET ('Value1', 'Value2', ...)	A field that can contain up to 64 predefined string values. SET data can be comma-delimited text referring to many values, or as an integer that is the bit representation of the values.
DATETIME	Stores a MySQL date and time in YYYY-MM-DD HH:MM:SS format. Roughly equivalent to a Lasso date string, but with a different format.
TIMESTAMP	MySQL time stamp for modification date.
DATE	Stores a MySQL date string in YYYY-MM-DD format.
TIME	Stores a MySQL time string in HH:MM:SS format.
YEAR	Efficient storage for four digit years. Rarely used.

To create a field:

Use the [Database_CreateField] tag to create a new field. The field will be inserted as the last field in the specified table.

- The following example shows two fields First_Name and Last_Name added to the People table of the Contacts database. Both fields are set to the data type VARCHAR with a maximum length of 64 characters.

```
[Database_CreateField: -Database='Contacts', -Table='People',  
-Field='First_Name', -Type='VARCHAR(64)']  
[Database_CreateField: -Database='Contacts', -Table='People',  
-Field='Last_Name', -Type='VARCHAR(64)']
```

- The following example shows a field Amount_Due being added to the People table. The field will store DECIMAL values with up to 14 digits and a precision of 2. This is a good data type for dollar amounts (up to \$999,999,999.99).

```
[Database_CreateField: -Database='Contacts', -Table='People',  
-Field='Amount_Due', -Type='DECIMAL(14,2)']
```

- The following example shows a field Notes being added to the People table. The field can store TEXT values up to 64k worth of text.

```
[Database_CreateField: -Database='Contacts', -Table='People',  
-Field='Notes', -Type='TEXT']
```

- The following example shows a field Job being added to the People table. The field can store one ENUM value selected from a list of four allowed values (Sales, Support, Management, or Engineering).

```
[Database_CreateField: -Database='Contacts', -Table='People',  
-Field='Job', -Type='ENUM('Sales', 'Support', 'Management', 'Engineering)']
```

To create a field in an existing table:

A field can be created in an existing table by using the -AfterField or -BeforeFirst parameters to the [Database_CreateField] tag. The order of fields in a database is not generally important, but it can be easier to use command line tools if the fields print out in a specific order.

- The following example shows a field Phone_Number being added to the Phone_Book table immediately after the Last_Name field. The field is defined as a fixed length CHAR data type which can store up to 16 digits.

```
[Database_CreateField: -Database='Example', -Table='Phone_Book',  
-Field='Phone_Number', -Type='CHAR(16)', -AfterField='Last_Name']
```

- The following example shows a field Title being added to the Phone_Book table before all other fields in the table. The field is defined as a fixed length CHAR data type which can store up to 8 characters.

```
[Database_CreateField: -Database='Example', -Table='Phone_Book',
-Field='Title', -Type='CHAR(8)', -BeforeFirst]
```

Note: Perform an [Inline] ... [/Inline] database action after creating a new field in order to force Lasso Administration to refresh and update its stored database information.

To change a field:

A field can be changed using the [Database_ChangeField] tag. This tag accepts all the same parameters as [Database_CreateField] with the addition of an -Original parameter that specifies the field to be changed. All of the parameters of the new field should be specified including the required name and type, any parameters left unspecified will be returned to their default values.

When a field is changed all the data in the field is translated to the new field type. Be sure to only change fields to compatible data types, otherwise there is a potential for data loss. If a field is changed to a smaller data type then any excess data beyond the size of the new data type will be lost.

The following example shows the Notes field from the Phone_Book table being changed so that it will only store about 255 characters in a TINYTEXT data type. Any characters beyond 255 in the records of Phone_Book will be truncated to 255 characters.

```
[Database_ChangeField: -Database='Example', -Table='Phone_Book',
-Original='Notes', -Field='Notes', -Type='TINYTEXT']
```

To remove a field:

Use the [Database_RemoveField] tag to drop the specified field from its table. This will eliminate all data stored in the field. The following example will remove a field named Title from the Phone_Book table.

```
[Database_RemoveField: -Database='Example', -Table='Phone_Book',
-Field='Title']
```

Optimizing Tables

After adding, changing, or removing many fields within a table it is good practice to optimize the table. This will ensure that the indices are up to date and that MySQL or Lasso MySQL has updated all of its internal information about the table.

Please see *Chapter 9: Administration Utilities* of the Lasso Professional 7 Setup Guide for more information about optimizing tables and automating database maintenance.

To optimize a table:

Use [Inline] ... [/Inline] tags with a -SQL command that specifies the OPTIMIZE TABLE and ANALYZE TABLE SQL statements. The following example optimizes the Phone_Book table of the Example database.

```
[Inline: -Database='Example', -SQL='OPTIMIZE TABLE Example.Phone_Book'][/Inline]
```

```
[Inline: -Database='Example', -SQL='ANALYZE TABLE Example.Phone_Book'][/Inline]
```

10

Chapter 10

FileMaker Pro Data Sources

This chapter documents tags and behaviors which are specific to FileMaker Pro data sources accessed using Lasso Connector for FileMaker Pro.

- *Overview* introduces FileMaker Pro data sources.
- *Performance Tips* includes recommendations which will help ensure that FileMaker Pro is used to its full potential.
- *Compatibility Tips* includes recommendations which help ensure that FileMaker Pro databases can be transferred to a different back-end data source.
- *FileMaker Pro Tags* describes tags specific to FileMaker Pro data sources.
- *Primary Key Field and Record ID* describes how the built-in record IDs in FileMaker Pro can be used as primary key fields.
- *Sorting Records* describes how custom sorts can be performed in FileMaker Pro databases.
- *Displaying Data* describes methods of returning field values from FileMaker Pro databases including repeating field values and values from portals.
- *Value Lists* describes how to retrieve and format value list data from FileMaker Pro databases.
- *Images* describes how to retrieve images from FileMaker Pro databases.
- *FileMaker Pro Scripts* describes how to activate FileMaker Pro scripts in concert with a Lasso database action.

Overview

Lasso Professional 7 allows access to FileMaker Pro data sources through Lasso Connector for FileMaker Pro. Connections can be made to any version of FileMaker Pro that includes Web Companion including FileMaker Pro 4.x and FileMaker Pro 5.x and 6.x Unlimited.

Please see *Chapter 7: Setting Up Data Sources* in the Lasso Professional 7 Setup Guide for information about how to configure FileMaker Pro and the Web Companion for access through Lasso Professional 7.

Lasso Connector for FileMaker Pro cannot access databases hosted by FileMaker Server directly. All databases must be opened and shared by a copy of the FileMaker Pro client. FileMaker Pro 3 is not supported since it does not include the Web Companion. Solutions built using FileMaker Developer which rely on a runtime engine are not supported.

LDML is a predominantly data source-independent language. It does include many FileMaker Pro specific tags which are documented in this chapter. However, all of the common procedures outlined in *Chapter 6: Data Source Fundamentals*, *Chapter 7: Searching and Displaying Data*, and *Chapter 8: Adding and Updating Records* can be used with FileMaker Pro data sources.

Note: The tags and procedures defined in this chapter can only be used with FileMaker Pro data sources. Any solution which relies on these tags cannot be easily retargeted to work with a different back-end database.

Terminology

Since Lasso works with many different data sources this documentation uses data source agnostic terms to refer to databases, tables, and fields. The following terms which are used in the FileMaker Pro documentation are equivalent to their Lasso counterparts.

- **Database** – Database is used to refer to a single FileMaker Pro database file. FileMaker Pro databases differ from other databases in Lasso in that they contain layouts rather than individual data tables. From a data storage point of view, a FileMaker Pro database is equivalent to a single Lasso MySQL table.
- **Layout** – Within Lasso a FileMaker Pro layout is treated as equivalent to a **Table**. The two terms can be used interchangeably. This equivalence simplifies Lasso security and makes transitioning between back-end data sources easier. All FileMaker Pro layouts can be thought of as views of a single data table. Lasso can only access fields which are contained in the layout named within the current database action.

- **Record** – FileMaker Pro records are referenced using a single -KeyValue rather than a -KeyField and -KeyValue pair. The -KeyField in FileMaker Pro is always the record ID which is set internally.
- **Fields** – The value for any field in the current layout in FileMaker Pro can be returned including the values for related fields, repeating fields, and fields in portals.

Although the equivalence of FileMaker Pro databases to Lasso MySQL databases and FileMaker Pro layouts to Lasso MySQL tables is imperfect, it is an essential compromise in order to map both database models onto Lasso Professional's two-tier (e.g. database and table) security model.

Performance Tips

This section contains a number of tips which will help get the best performance from a FileMaker Pro database. Since queries must be performed sequentially within FileMaker Pro, even small optimizations can yield significant increases in the speed of Web serving under heavy load.

- **Dedicated FileMaker Pro Machine** – For best performance, place the FileMaker Pro client on a different machine from Lasso Service and the Web server application.
- **FileMaker Pro Server** – If a FileMaker Pro database must be accessed by a mix of FileMaker Pro clients and Web visitors through Lasso, it should be hosted on FileMaker Pro Server. Lasso will access the database through a single FileMaker Pro client which is connected as a guest to FileMaker Pro Server.
- **Web Companion** – Always ensure that the latest version of FileMaker Pro Web Companion for the appropriate version of FileMaker Pro is installed.
- **Index Fields** – Any fields which will be searched through Lasso should have indexing turned on. Avoid searching on unstored calculation fields, related fields, and summary fields.
- **Custom Layouts** – Layouts should be created with the minimal number of fields required for Lasso. All the data for the fields in the layout will be sent to Lasso with the query results. Limiting the number of fields can dramatically cut down the amount of data which needs to be sent from FileMaker Pro to Lasso.
- **Return Fields** – Use the -ReturnField tag to limit the number of fields which are returned to Lasso. If no -ReturnField tag is specified then all of the data for the fields in the current layout will be sent to Lasso with the query results.

- **Sorting** – Sorting can have a serious impact on performance if large numbers of records must be sorted. Avoid sorting large record sets and avoid sorting on calculation fields, related fields, unindexed fields, or summary fields.
- **Contains Searching** – FileMaker Pro is optimized for the default Begins With searches (and for numerical searches). Use of the contains operator `cn` can dramatically slow down performance since FileMaker Pro will not be able to use its indices to optimize searches.
- **Max Records** – Using `-MaxRecords` to limit the number of records returned in the result set from FileMaker Pro can speed up performance. Use `-MaxRecords` and `-SkipRecords` or the `[Link_...]` tags to navigate a visitor through the found set.
- **Calculation Fields** – Calculation fields should be avoided if possible. Searching or sorting on unindexed, unstored calculation fields can have a negative effect on FileMaker Pro performance.
- **FileMaker Pro Scripts** – The use of FileMaker Pro scripts should be avoided if possible. When FileMaker Pro executes a script, no other database actions can be performed at the same time. FileMaker Pro scripts can usually be rewritten as LDML to achieve the same effect as the script, often with greater performance.

In addition to these tips, Lasso MySQL can be used to shift some of the burden off of FileMaker Pro. Lasso MySQL can usually perform database searches much faster than FileMaker Pro. Lasso Professional 7 also includes sessions and compound data types that can be used to perform some of the tasks of a database, but with higher performance for small amounts of data.

Compatibility Tips

Following these tips will help to ensure that it is easy to transfer data from a FileMaker Pro database to another data source, such as the built-in Lasso MySQL database, at a future date.

- **Database Names** – Database, layout, and field names should contain only a mix of letters, numbers, and the underscore character. They should not contain any punctuation other than spaces.
- **Calculation Fields** – Avoid the use of calculation fields. Instead, perform calculations within Lasso and store the results back into regular fields if they will be needed later.
- **Summary Fields** – Avoid the use of summary fields. Instead, summarize data using `[Inline]` searches within Lasso.

- **Scripts** – Avoid the use of FileMaker Pro scripts. Most actions which can be performed with scripts can be performed using the database actions available within Lasso.
- **Record ID** – Create a calculation field named ID and assign it to the following calculation. Always use the -KeyField='ID' within [Inline] database actions, HTML forms, and URLs. This ensures that when moving to a database that relies on storing the key field value explicitly, a unique key field value is available.

```
Status(CurrentRecordID)
```

FileMaker Pro Tags

LDML 7 includes tags that allow the type of a database to be inspected.

Table 1: FileMaker Pro Data Source Tag

Tag	Description
[Lasso_DataSourcesFileMaker]	Returns True if the specified database is hosted by FileMaker Pro.

To check whether a database is hosted by FileMaker Pro:

The following example shows how to use [Lasso_DataSourcesFileMaker] to check whether or not the database Example is hosted by FileMaker Pro.

```
[If: (Lasso_DataSourcesFileMaker: 'Example.fp5')]
  Example is hosted by FileMaker Pro!
[Else]
  Example is not hosted by FileMaker Pro.
[/If]
```

→ Example is hosted by FileMaker Pro!

To list all databases hosted by FileMaker Pro:

Use the [Database_Names] ... [/Database_Names] tags to list all databases available to Lasso. The [Lasso_DataSourcesFileMaker] tag can be used to check each database and only those that are hosted by FileMaker Pro will be returned. The result shows two databases, Contacts.fp5 and Example.fp5, which are available through FileMaker Pro.

```
[Database_Names]
  [If: (Lasso_DataSourcesFileMaker:(Database_NameItem))]
    <br>[Database_NameItem]
  [/If]
[/Database_Names]
```

→
Example.fp5

Contacts.fp5

Primary Key Field and Record ID

FileMaker Pro databases include a built-in primary key value called the Record ID. This value is guaranteed to be unique for any record in a FileMaker Pro database. It is predominantly sequential, but should not be relied upon to be sequential. The values of the record IDs within a database may change after an import or after a database is compressed using Save a Copy As.... Record IDs can be used within a solution to refer to a record on multiple pages, but should not be stored as permanent references to FileMaker Pro records.

Note: The tag [RecordID_Value] can also be used to retrieve the record ID from FileMaker Pro records. However, for best results, it is recommended that the [KeyField_Value] tag be used.

To return the current record ID:

The record ID for the current record can be returned using [KeyField_Value]. The following example shows [Inline] ... [/Inline] tags that perform a -FindAll action and return the record ID for each returned record using the [KeyField_Value] tag.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -FindAll]
  [Records]
    <br>[KeyField_Value]: [Field: 'First_Name'] [Field: 'Last_Name']
  [/Records]
[/Inline]
```

→
126: John Doe

127: Jane Doe

4096: Jane Person

To reference a record by record ID:

For -Update and -Delete command tags the record ID for the record which should be operated upon can be referenced using -KeyValue. The -KeyField does not need to be defined or should be set to an empty string if it is, -KeyField=".

- The following example shows a record in Contacts.fp5 being updated with -KeyValue=126. The name of the person referenced by the record is changed to John Surname.

```
[Inline: -Database='Contacts.fp5',
-Layout='People',
-KeyValue=126,
'First_Name'='John',
'Last_Name'='Surname',
-Update]
<br>[KeyField_Value]: [Field: 'First_Name'] [Field: 'Last_Name']
[/Inline]
```

→
126: John Surname

- The following example shows a record in Contacts.fp5 being deleted with -KeyValue=127. The -KeyField command tag is included, but its value is set to the empty string.

```
[Inline: -Database='Contacts.fp5',
-Layout='People',
-KeyField='',
-KeyValue=126,
-Delete]
[/Inline]
```

To access the record ID within FileMaker Pro:

The record ID for the current record in FileMaker Pro can be accessed using the calculation value Status(CurrentRecordID) within FileMaker Pro.

Sorting Records

In addition to the Ascending and Descending values for the -SortOrder tag, FileMaker Pro data sources can also accept a Custom value. The Custom value can be used for any field which is formatted with a value list in the current layout. The field will be sorted according to the order of values within the value list.

To return custom sorted results:

Specify -SortField and -SortOrder command tags within the search parameters. The following [Inline] ... [/Inline] tags include sort command tags specified in hidden inputs. The records are first sorted by title in custom order, then by Last_Name and First_Name in ascending order. The Title field will be sorted in the order of the elements within the value list associated with the field in the database. In this case, it will be sorted as Mr., Mrs., Ms.

```
[Inline: -FindAll,
-Database='Contacts.fp5',
-Table='People',
-KeyField='ID',
```

```
-SortField='Title', -SortOrder='Custom',  
-SortField='Last_Name', -SortOrder='Ascending',  
-SortField='First_Name', -SortOrder='Ascending']  
[Records]  
<br>[Field: 'Title'] [Field: 'First_Name'] [Field: 'Last_Name']  
[/Records]  
[/Inline]
```

The following results could be returned when this page is loaded. Each of the records with a title of Mr. appear before each of the records with a title of Mrs. Within each title, the names are sorted in ascending alphabetical order.

→
Mr. John Doe

Mr. John Person

Mrs. Jane Doe

Mrs. Jane Person

Displaying Data

FileMaker Pro includes a number of container tags and substitution tags that allow the different types of FileMaker Pro fields to be displayed. These tags are summarized in *Table 2: FileMaker Pro Data Display Tags* and then examples are included in the sections that follow.

See also the sections on *Value Lists* and *Images* for more information about returning values from FileMaker Pro fields.

Table 2: FileMaker Pro Data Display Tags

Tag	Description
[Field]	Can be used to reference FileMaker Pro fields including related fields and repeating fields.
[Repeating] ... [/Repeating]	Container tag repeats for each defined repetition of a repeating field. Requires a single parameter, the name of the repeating field from the current layout.
[Repeating_Valueltem]	Returns the value for each repetition of a repeating field.
[Portal] ... [/Portal]	Container tag repeats for each record in a portal. Requires a single parameter, the name of the portal relationship from the current layout.

Note: All fields which are referenced by Lasso must be contained in the current layout in FileMaker Pro. For portals and repeating fields only the number of repetitions shown in the current layout will be available to Lasso.

Related Fields

Related fields are named using the relationship name followed by two colons :: and the field name. For example, a related field `Call_Duration` from a `Calls.fp5` database might be referenced as `Calls.fp5::Call_Duration`. Any related fields which are included in the layout specified for the current Lasso action can be referenced using this syntax. Data can be retrieved from related fields or it can be set in related fields when records are added or updated.

To return data from a related field:

Specify the name of the related field within a `[Field]` tag. The related field must be contained in the current layout either individually or within a portal. In a one-to-one relationship, the value from the single related record will be returned. In a one-to-many relationship, the value from the first related record as defined by the relationship options will be returned. See the section on *Portals* below for more control over one-to-many relationships.

The following example shows a `-FindAll` action being performed in a database `Contacts.fp5`. The related field `Last_Call_Time` from the `Calls.fp5` databases is returned for each record through a relationship named `Calls.fp5`.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -FindAll]
[Records]
  <br>[KeyField_Value]: [Field: 'First_Name'] [Field: 'Last_Name']
  (Last call at: [Field: 'Calls::Last_Call_Time']).
[/Records]
[/Inline]
```

```
→ <br>126: John Doe (Last call at 12:00 pm).
   <br>127: Jane Doe (Last call at 9:25 am).
   <br>4096: Jane Person (Last call at 4:46 pm).
```

To set the value for a related field:

Specify the name of the related field within the action which adds or updates a record within the database. The related field must be contained in the current layout either individually or within a portal. In a one-to-one relationship, the value for the field in a single related record will be modified. In a one-to-many relationship, the value for the field in the first related record as defined by the relationship options will be modified. See the section on *Portals* below for more control over one-to-many relationships.

The following example shows an `-Update` action being performed in a database `Contacts.fp5`. The related field `Last_Call_Time` from the `Calls.fp5` database is updated for Jane Person. The new value is returned.

```
[Inline: -Database='Contacts.fp5',
-Layout='People',
-KeyValue=4096,
'Calls.fp5::Last_Call_Time'='1:45 am',
-Update]
<br>[KeyField_Value]: [Field: 'First_Name'] [Field: 'Last_Name']
(Last call at: [Field: 'Calls.fp5::Last_Call_Time']).
[/Inline]
```

→
4096: Jane Person (Last call at 1:45 pm).

Portals

Portals allow one-to-many relationships to be displayed within FileMaker Pro databases. Portals allow data from many related records to be retrieved and displayed in a single format file. A portal must be present in the current FileMaker Pro layout in order for its values to be retrieved using Lasso.

Only the number of repetitions formatted to display within FileMaker Pro will be displayed using Lasso. A portal must contain a scroll bar in order for all records from the portal to be displayed using Lasso.

Fields in portals are named using the same convention as related fields. The relationship name is followed by two colons :: and the field name. For example, a related field `Call_Duration` from a `Calls.fp5` database might be referenced as `Calls.fp5::Call_Duration`.

Note: Everything that is possible to do with portals can also be performed using nested `[Inline] ... [/Inline]` tags to perform actions in the related database. Portals are unique to FileMaker Pro databases.

To return values from a portal:

Use the `[Portal] ... [/Portal]` tags with the name of the portal referenced in the opening `[Portal]` tag. `[Field]` tags within the `[Portal] ... [/Portal]` tags should reference the fields from the current portal row using related field syntax.

The following example shows a portal `Calls.fp5` that is contained in the `People` layout of the `Contacts.fp5` database. The Time, Duration, and Number of each call is displayed.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -FindAll]
[Records]
<p>Calls for [Field: 'First_Name'] [Field: 'Last_Name']:
[Portal: 'Calls.fp5']
<br>[Field: 'Calls.fp5::Number'] at [Field: 'Calls.fp5::Time']
```



```

        for [Field: 'Calls.fp5::Duration'] minutes.
    [/Portal]
[/Records]
[/Inline]
→ <p>Calls for John Doe:
  <br>555-1212 at 12:00 pm for 15 minutes.

  <p>Calls for Jane Doe:
  <br>555-1212 at 9:25 am for 60 minutes.

  <p>Calls for Jane Person:
  <br>555-1212 at 2:23 pm for 55 minutes.
  <br>555-1212 at 4:46 pm for 5 minutes.

```

To add a record to a portal:

A record can be added to a portal by adding the record directly to the related database. In the following example the Calls.fp5 database is related to the Contacts.fp5 database by virtue of a field Contact_ID that stores the ID for the contact which the calls were made to. New records added to Calls.fp5 with the appropriate Contact_ID will be shown through the portal to the next site visitor.

In the following example a new call is added to the Calls.fp5 database for John Doe. John Doe has an ID of 123 in the Contacts.fp5 database. This is the value used for the Contact_ID field in Calls.fp5.

```

[Inline: -Add,
  -Database='Calls.fp5',
  -Layout='People',
  'Contact_ID'=123,
  'Number'='555-1212',
  'Time'='12:00 am',
  'Duration'=55]
[/Inline]

```

To update a record within a portal:

In order to update records shown within a portal it is recommended that you use a field to return the record ID of each record in the portal, then use that value in nested [Inline] ... [/Inline] tags to update the related record.

Create a calculation field named RecordID within the related database (e.g. Calls.fp5) that contains the following FileMaker Pro calculation.

```
Status(CurrentRecordID)
```

Place that field within the portal shown within the main database (e.g. Contacts.fp5). To perform an update of a portal row, use [Inline] ... [/Inline] tags which reference the related database and the RecordID from the portal.

The following example shows how to update every record contained within a portal. The field `Approved` is set to `Yes` for each call from the `Calls.fp5` database for all contacts from the `Contacts.fp5` database.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -FindAll]
[Records]
[Portal: 'Calls.fp5']
[Inline: -Database='Calls.fp5',
  -Layout='People',
  -KeyField=(Field: 'Calls.fp5::RecordID'),
  'Approved'='Yes',
  -Update]
[/Inline]
[/Portal]
[/Records]
[/Inline]
```

The results of the action will be shown the next time the portal is viewed by a site visitor.

To delete a record from a portal:

The same method as described above for updating records within a portal can be used to delete records from a portal. In the following example, all records from `Contacts.fp5` are returned and every record from the `Calls.fp5` portal is deleted.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -FindAll]
[Records]
[Portal: 'Calls.fp5']
[Inline: -Database='Calls.fp5',
  -Layout='People',
  -KeyField=(Field: 'Calls.fp5::RecordID'),
  -Delete]
[/Inline]
[/Portal]
[/Records]
[/Inline]
```

No records will be contained in the portal the next time the site is viewed by a site visitor. However, not all records in `Calls.fp5` have necessarily been deleted. Any records which were not associated with a contact in `Contacts.fp5` will still remain in the database.

Repeating Fields

Repeating fields in FileMaker Pro allow many values to be stored in a single field. Each repeating field is defined to hold a certain number of values. These values can be retrieved using the tags defined in this section.

See the documentation for FileMaker Pro for more information about how to create and use repeating fields within FileMaker Pro.

In order to display or set values in a repeating field, the layout referenced in the current database action must contain the repeating field formatted to show the desired number of repetitions. If a field is set to store eight repetitions, but only to show two, then it will appear to be a two-repetition field to Lasso.

Note: The use of repeating fields is not recommended. Usually a simple text field which contains multiple values separated by returns can be used for the same effect through Lasso. For more complex solutions a related database and [Portal] ... [/Portal] tags or nested [Inline] ... [/Inline] tags can often be easier to use and maintain than a solution with repeating fields.

To return values from a repeating field:

Use the [Repeating] ... [/Repeating] and [Repeating_ValueItem] tags to return each of the values from a repeating field. The opening [Repeating] tag takes a single parameter which names a field from the current FileMaker Pro layout that repeats. The contents of the [Repeating] ... [/Repeating] tags is repeated for each repetition and the [Repeating_ValueItem] tag is used to return the value for the current repetition.

The following example shows a repeating field Customer_ID that has four repetitions. Normally, only the first repetition has a defined value, but for a contact that has multiple accounts, multiple values are defined. Since Jane Person has two customer accounts, two repetitions of Customer_ID are returned.

```
[Inline: -Database='Contacts', -Layout='People', 'Last_Name'='Person', -Search]
[Records]
  <p>[Field: 'First_Name'] [Field: 'Last_Name']
  [Repeating: 'Customer_ID']
    <br>Customer ID [Loop_Count]: [Repeating_ValueItem].
  [/Repeating]
[/Records]
[/Inline]
```

```
→ <p>Jane Person
  <br>Customer ID 1: 100123.
  <br>Customer ID 2: 123654.
```

To add a record with a repeating field:

A record can be added with values in a repeating field by referencing the field multiple times within the -Add action. The following example shows a new contact being added to Contacts.fp5. The contact Jimmy Last_Name is given three customer ID numbers referenced by the field Customer_ID

multiple times. The added record is returned showing all three customer IDs are stored.

```
[Inline: -Database='Contacts',
-Layout='People',
'First_Name'='Jimmy',
'Last_Name'='Last_Name',
'Customer_ID'='2001',
'Customer_ID'='2010',
'Customer_ID'='2061',
-Add]
<p>[Field: 'First_Name'] [Field: 'Last_Name']
[Repeating: 'Customer_ID']
<br>Customer ID [Loop_Count]: [Repeating_Valueltem].
[/Repeating]
[/Inline]
```

→ <p>Jimmy Last_Name

Customer ID 1: 2001.

Customer ID 2: 2010.

Customer ID 3: 2061.

To update a record with a repeating field:

A repeating field can be updated by referencing it multiple times within the -Update action. The following example shows an HTML form which displays four repetitions of the field Customer_ID and allows each of them to be modified. Notice that the four repetitions are created using the looping [Repeating] ... [/Repeating] container tags.

```
<form action="response.lasso" method="POST">
<input type="hidden" name="-Database" value="Contacts.fp5">
<input type="hidden" name="-Layout" value="People">
<input type="hidden" name="-KeyValue" value="[KeyField_Value]">

<p>First Name:
<input type="text" name="First_Name" value="[Field: 'First_Name']">
<br>Last Name:
<input type="text" name="Last_Name" value="[Field: 'Last_Name']">

[Repeating: 'Customer ID']
<br>Customer ID:
<input type="text" name="Customer_ID" value="[Repeating_Valueltem]">
[/Repeating]

<p><input type="submit" name="-Update" value="Update this Record">
</form>
```

To delete values from a repeating field:

- Records which contain repeating fields can be deleted using the same technique for deleting any FileMaker Pro records. All repetitions of the repeating field will be deleted along with the record. The following [Inline] ... [/Inline] tags will delete the record with a record ID of 127.

```
[Inline: -Database='Contacts.fp5', -Table='People', -KeyValue=127, -Delete]
<p>The record was deleted.
[/Inline]
```

- A single repetition of a repeating field can be deleted by setting its value to an empty string. The other values in the repeating field will not slide down to fill in the missing repetition. The following [Inline] ... [/Inline] will set the first repetition of a repeating field Customer_ID to the empty string, but leave the second and third repetitions unchanged.

The values for the repeating field are first placed in an array so that they can be referenced by number within the opening [Inline] tag.

```
[Variable: 'Customer_ID' = (Array: ", ", ")]
[Repeating: 'Customer_ID']
  [(Variable: 'Customer_ID')->(Get: Loop_Count) = (Repeating_Valueltem)]
[/Repeating]

[Inline: -Update,
  -Database='Contacts.fp5',
  -Table='People',
  -KeyValue=127,
  'Customer_ID'="",
  'Customer_ID'=(Variable: 'Customer_ID')->(Get: 2),
  'Customer_ID'=(Variable: 'Customer_ID')->(Get: 3),
  <p>[Field: 'First_Name'] [Field: 'Last_Name']
  [Repeating: 'Customer_ID']
    <br>Customer ID [Loop_Count]: [Repeating_Valueltem].
  [/Repeating]
[/Inline]
```

The results show that the value for the first repetition of the repeating field has been deleted, but the second and third repetitions remain intact.

```
→ <p>Jimmy Last_Name
  <br>Customer ID 1: .
  <br>Customer ID 2: 2010.
  <br>Customer ID 3: 2061.
```

Value Lists

Value lists in FileMaker Pro allow a set of possible values to be defined for a field. The items in the value list associated with a field on the current layout for a Lasso action can be retrieved using the tags defined in *Table 3: FileMaker Pro Value List Tags*. See the documentation for FileMaker Pro for more information about how to create and use value lists within FileMaker Pro.

In order to display values from a value list, the layout referenced in the current database action must contain a field formatted to show the desired value list as a pop-up menu, select list, check boxes, or radio buttons. Lasso cannot reference a value list directly. Lasso can only reference a value list through a formatted field in the current layout.

Table 3: FileMaker Pro Value List Tags

Tag	Description
[Value_List] ... [/Value_List]	Container tag repeats for each value in the named value list. Requires a single parameter, the name of a field from the current layout which has a value list assigned to it.
[Value_ListItem]	Returns the value for the current item in a value list. Optional -Checked or -Selected parameter returns only currently selected values from the value list.
[Selected]	Displays the word Selected if the current value list item is selected in the field associated with the value list.
[Checked]	Displays the word Checked if the current value list item is selected in the field associated with the value list.
[Option]	Generates a series of <option> tags for the value list. Requires a single parameter, the name of a field from the current layout which has a value list assigned to it.

Note: See *Chapter 7: Searching and Displaying Data* for information about the -Show command tag which is used throughout this section.

To display all values from a value list:

- The following example shows how to display all values from a value list using a -Show action within [Inline] ... [/Inline] tags. The field Title in the Contacts.fp5 database contains five values Mr., Mrs., Ms., and Dr. The -Show action allows the values for value lists to be retrieved without performing a database action.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -Show]
[Value_List: 'Title']
  <br>[Value_ListItem]
[/Value_List]
[/Inline]
```

```
→ <br>Mr.
   <br>Mrs.
   <br>Ms.
   <br>Dr.
```

- The following example shows how to display all values from a value list using a named inline. The same name Values is referenced by -InlineName in both the [Inline] tag and [Value_List] tag.

```
[Inline: -InlineName='Values', -Database='Contacts.fp5', -Layout='People', -Show]
[/Inline]
...
[Value_List: 'Title', -InlineName='Values']
  <br>[Value_ListItem]
[/Value_List]
```

```
→ <br>Mr.
   <br>Mrs.
   <br>Ms.
   <br>Dr.
```

To display an HTML pop-up menu in an -Add form with all values from a value list:

- The following example shows how to format an HTML `<select> ... </select>` pop-up menu to show all the values from a value list. A select list can be created with the same code by including size and/or multiple parameters within the `<select>` tag. This code is usually used within an HTML form that performs an -Add action so the visitor can select a value from the value list for the record they create.

The example shows a single `<select> ... </select>` within [Inline] ... [/Inline] tags with a -Show command. If many value lists from the same database are being formatted, they can all be contained within a single set of [Inline] ... [/Inline] tags.

```
<form action="response.lasso" method="POST">
  <input type="hidden" name="-Add" value="">
  <input type="hidden" name="-Database" value="Contacts.fp5">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
```

```

[Inline: -Database='Contacts.fp5', -Layout='People', -Show]
  <select name="Title">
    [Value_List: 'Title']
    <option value="[Value_ListItem]">[Value_ListItem]</option>
  [/Value_List]
</select>
[/Inline]

<p><input type="submit" name="-Add" value="Add Record">
</form>

```

- The [Option] tag can be used to easily format a value list as an HTML <select> ... </select> pop-up menu. The [Option] tag generates all of the <option> ... </option> tags for the pop-up menu based on the value list for the specified field. The example below generates exactly the same HTML as the example above.

```

<form action="response.lasso" method="POST">
  <input type="hidden" name="-Add" value="">
  <input type="hidden" name="-Database" value="Contacts.fp5">
  <input type="hidden" name="-Table" value="People"?
  <input type="hidden" name="-KeyField" value="ID">

[Inline: -Database='Contacts.fp5', -Layout='People', -Show]
  <select name="Title">
    [Option: 'Title']
  </select>
[/Inline]

<p><input type="submit" name="-Add" value="Add Record">
</form>

```

To display HTML radio buttons with all values from a value list:

The following example shows how to format a set of HTML <input> tags to show all the values from a value list as radio buttons. The visitor will be able to select one value from the value list. Check boxes can be created with the same code by changing the type from radio to checkbox.

```

<form action="response.lasso" method="POST">
  <input type="hidden" name="-Add" value="">
  <input type="hidden" name="-Database" value="Contacts.fp5">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">

[Inline: -Database='Contacts.fp5', -Layout='People', -Show]
  [Value_List: 'Title']
  <input type="radio" name="Title" value="[Value_ListItem]"> [Value_ListItem]
[/Value_List]
[/Inline]

```



```
<p><input type="submit" name="-Add" value="Add Record">
</form>
```

To display only selected values from a value list:

The following examples show how to display the selected values from a value list for the current record. The record for John Doe is found within the database and the selected value for the Title field, Mr. is displayed.

- The -Selected keyword in the [Value_ListItem] tag ensures that only selected value list items are shown. The following example uses a conditional to check whether [Value_ListItem: -Selected] is empty.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -KeyValue=126, -Search]
[Value_List: 'Title']
  [If: (Value_ListItem: -Selected) != ""]
    <br>[Value_ListItem: -Selected]
  [/If]
[/Value_List]
[/Inline]
```

→
Mr.

- The [Selected] tag ensures that only selected value list items are shown. The following example uses a conditional to check whether [Selected] is empty and only shows the [Value_ListItem] if it is not.

```
[Inline: -Database='Contacts.fp5', -Layout='People', -KeyValue=126, -Search]
[Value_List: 'Title']
  [If: (Selected) != ""]
    <br>[Value_ListItem]
  [/If]
[/Value_List]
[/Inline]
```

→
Mr.

- The [Field] tag can also be used simply to display the current value for a field without reference to the value list.

```
<br>[Field: 'Title']
```

→
Mr.

To display an HTML popup menu in an -Update form with selected value list values:

- The following example shows how to format an HTML `<select> ... </select>` select list to show all the values from a value list with the selected values highlighted. The [Selected] tag returns Selected if the current value list item is selected in the database or nothing otherwise. This code will usually be used in an HTML form that performs an -Update

action to allow the visitor to see what values are selected in the database currently and make different choices for the updated record.

```
<form action="response.lasso" method="POST">
  <input type="hidden" name="-Update" value="">
  <input type="hidden" name="-Database" value="Contacts.fp5">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-KeyValue" value="127">

  [Inline: -Database='Contacts.fp5', -Layout='People', -KeyValue=126, -Search]
  <select name="Title" multiple size="4">
    [Value_List: 'Title']
    <option value="[Value_ListItem]" [Selected]>[Value_ListItem]</option>
  [/Value_List]
</select>
[/Inline]

  <p><input type="submit" name="-Update" value="Update Record">
</form>
```

- The [Option] tag automatically inserts Selected parameters as needed to ensure that the proper options are selected in the HTML select list. The example below generates exactly the same HTML as the example above.

```
<form action="response.lasso" method="POST">
  <input type="hidden" name="-Update" value="">
  <input type="hidden" name="-Database" value="Contacts.fp5">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-KeyValue" value="127">

  [Inline: -Database='Contacts.fp5', -Layout='People', -KeyValue=126, -Search]
  <select name="Title" multiple size="4">
    [Option: 'Title']
  </select>
[/Inline]

  <p><input type="submit" name="-Update" value="Update Record">
</form>
```

To display HTML check boxes with selected value list values:

The following example shows how to format a set of HTML <input> tags to show all the values from a value list as check boxes with the selected check boxes checked. The [Checked] tag returns Checked if the current value list item is selected in the database or nothing otherwise. Radio buttons can be created with the same code by changing the type from checkbox to radio.

```

<form action="response.lasso" method="POST">
  <input type="hidden" name="-Update" value="">
  <input type="hidden" name="-Database" value="Contacts.fp5">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-KeyValue" value="127">

[Inline: -Database='Contacts.fp5', -Layout='People', -KeyValue=126, -Search]
[Value_List: 'Title']
  <input type="checkbox" name="Title" value="[Value_ListItem]" [Checked]>
  [Value_ListItem]
[/Value_List]
[/Inline]

<p><input type="submit" name="-Update" value="Update Record">
</form>

```

Images

Images can be served directly from FileMaker Pro container fields. Any image which can be stored in a container field can be served. Stored GIFs and JPEGs will be served straight from the database. All other image formats will first be converted to JPEG format and then served.

There are two ways to serve an image from a FileMaker Pro database. The [Image_URL] tag can be used to generate the URL for an image within an HTML tag. The [IMG] tag can be used to generate a whole tag complete with URL.

Table 4: Image Tags

Tag	Description
[Image_URL]	Returns the URL for an image from a FileMaker Pro database. Requires a single parameter, the name of the container field which contains the image.
[IMG]	Returns an HTML tag referencing the URL for an image from a FileMaker Pro database. Requires a single parameter, the name of the container field which contains the image.

To serve an image from a FileMaker Pro database using [Image_URL]:

The [Image_URL] tag can be used to return a URL that retrieves an image from the FileMaker Pro database. The [Image_URL] is used like a [Field] tag. Its only parameter is the name of the container field that contains the

image to be served. It should be placed in an HTML tag with optional size parameters.

The following code inserts the image stored in the Photo field for the John Doe record in the Contacts.fp5 database.

```
[Inline: -Database='Contacts.fp5',
  -Layout='People',
  'First_Name'='John',
  'Last_Name'='Doe',
  -Search]
[Records]
  
[/Records]
[/Inline]
```

The HTML generated by this example is a Lasso action which includes an -Image command tag to return the image from the container field in the FileMaker Pro database.

➔

Note: Additional parameters can be specified within the HTML tag in order to specify the width and height of the returned image. The image will be scaled to the desired size. See the next section for details.

[IMG] Tag

The [IMG] tag can be used to return an entire HTML tag. The URL of the referenced image is the same as that generated by the [Image_URL] tag. The optional parameters detailed in *Table 5: [IMG] Parameters* allow the various parameters of the HTML tag to be set.

Table 5: [IMG] Parameters

Parameter	Description
Value	The name of the field which contains the image. Required.
-IMGAlt	The alt parameter of the tag. Optional.
-IMGWidth	The width parameter of the tag. Optional.
-IMGHeight	The height parameter of the tag. Optional.
-IMGBorder	The border parameter of the tag. Optional.
-IMGIsMap	Specifies that the image is a map. Optional.
-IMGUseMap	The usemap parameter of the tag. Optional.
-IMGAlign	The align parameter of the tag. Optional.
-IMGName	The name parameter of the tag. Optional.
-IMGVSpace	The vspace parameter of the tag. Optional.
-IMGHSpace	The hspace parameter of the tag. Optional.
-IMGOptions	Allows other options to be added to the tag. Optional.

To serve an image from a FileMaker Pro database using [IMG]:

The [IMG] tag can be used to return an HTML tag with many optional parameters. The following code inserts the image stored in the Photo field for the John Doe record in the Contacts.fp5 database. The image is scaled so that its height is 72 pixels. It's width will be automatically adjusted by the Web browser so the image's dimensions stay in proportion.

```
[Inline: -Database='Contacts.fp5',
  -Layout='People',
  'First_Name'='John',
  'Last_Name'='Doe',
  -Search]
[Records]
  [IMG: 'Photo', -IMGHeight=72]
[/Records]
[/Inline]
```

The HTML generated by this example is a Lasso action which includes an -Image command tag to return the image from the container field in the FileMaker Pro database. Since the -IMGHeight parameter was included the resulting tag includes a height parameter.

➔

FileMaker Pro Scripts

LDML includes command tags which allow scripts in FileMaker Pro databases to be executed. Scripts are usually executed in concert with a database action. They can be performed before the database action, after the database action but before the results are sorted, or just before the results are returned to Lasso. The command tags for executing FileMaker Pro scripts are described in *Table 7: FileMaker Pro Scripts Tags*.

FileMaker Pro Tip: It is best to limit the use of FileMaker Pro scripts. Most functionality of FileMaker Pro scripts can be achieved in LDML with better performance especially on a busy Web server.

Table 7: FileMaker Pro Scripts Tags

Tag	Description
-FMScript	Specifies a script to be processed after the current database action has been performed. Requires a single parameter which names a FileMaker Pro script. Synonym is -FMScriptPost.
-FMScriptPre	Specifies a script to be processed before the current database action has been performed. Requires a single parameter which names a FileMaker Pro script.
-FMScriptPreSort	Specifies a script to be processed after the current database action, but before the results are sorted. Requires a single parameter which names a FileMaker Pro script.

Conditions for executing a FileMaker Pro script:

- 1 The script must be defined in the database referenced by the action in which the -FMScript... tag is called.
- 2 The current user must have permission to execute scripts. See the Group section in *Chapter 8: Setting Up Security* of the Lasso Professional 7 Setup Guide for more information.
- 3 The found set should not be empty after performing a FileMaker Pro script. Scripts should always ensure that they return a non-empty found set after they execute.
- 4 All database action on the FileMaker Pro machine must wait until the script finishes. Scripts should be as fast and efficient as possible.

To execute a FileMaker Pro script within [Inline] ... [/Inline] tags:

The following example shows a FileMaker Pro script named `Filter_People` being called after a `-FindAll` action is performed within a FileMaker Pro database `Contacts.fp5`. The script removes certain records from the found set and returns the results.

```
[Inline: -Database='Contacts.fp5',
  -Layout='People',
  -FMScript='Filter_People',
  -FindAll]
...
[/Inline]
```

The results of the `[Inline] ... [/Inline]` tags will be the result of the script `Filter_People`. The record set and its order can be completely determined by the script.

To execute a FileMaker Pro script within an HTML form:

The following example shows a FileMaker Pro script named `Clean_Up` being performed before a `-FindAll` action is performed within `Contacts.fp5`. The script deletes invalid records so that the found set will only contain valid records after the `-FindAll` is performed. The script is performed before the database action since it is called with `-FMScriptPre`.

```
<form action="response.lasso" method="POST">
  <input type="hidden" name="-FindAll">
  <input type="hidden" name="-Database" value="Contacts.fp5">
  <input type="hidden" name="-Layout" value="People">
  <input type="hidden" name="-FMScriptPre" value="Clean_Up">

  <br><input type="submit" name="-FindAll" value="Find All">
</form>
```

The results of the script include all valid records that were not deleted by the `Clean_Up` script.

To execute a FileMaker Pro script within a URL:

The following example shows a script named `Update_Priority` which is performed after the `-FindAll` database action, but before the results are sorted. The `Update_Priority` script could update a field `Priority`, based on the records from the current found set, which the sort depends on. The script is called using the `-FMScriptPreSort` tag.

```
<a href="response.lasso?-Database=Contacts.fp5&
  -Layout=People&
  -FMScriptPreSort=Update_Priority&
  -SortOrder=Descending&
```

```
-SortField=Priority&  
-FindAll">  
Find All and Sort by Priority  
</a>
```

The results of this URL, when it is selected, will be all records from the databases, sorted in descending order according to the value of the Priority field after it has been updated by the Update_Priority script.

➔ ``

Note: Additional parameters can be specified within the HTML `` tag in order to specify the width and height of the returned image. The image will be scaled to the desired size. See the next section for details.

11

Chapter 11

JDBC Data Sources

This chapter documents the usage of LDML 7 with JDBC data sources.

- *Overview* introduces JDBC data source support in Lasso Professional 7.
- *Using JDBC Data Sources* describes using JDBC data sources with Lasso Professional 7.
- *JDBC Schema Tags* describes using LDML tags to return schema values from JDBC data sources that support schema ownership.

Overview

Native support for JDBC data sources is included in Lasso Professional 7 in addition to native support for FileMaker Pro and MySQL data sources. If a JDBC driver is available for a data source, it can be installed to Lasso Professional 7, allowing Lasso to instantly communicate with that data source. This feature allows Lasso Professional 7 to communicate with over 150 JDBC-compliant data sources, including Sybase, DB2, Frontbase, Openbase, Interbase, and Microsoft SQL Server 2000. For more information on JDBC connectivity and availability for a particular data source, see the data source documentation or contact the data source manufacturer.

Lasso Professional 7 functions as its own JDBC driver manager, and all JDBC drivers must be installed directly to Lasso Professional 7. Instructions on how to set up a JDBC data source for use with Lasso Professional are documented in *Chapter 7: Setting Up Data Sources* in the Lasso Professional 7 Setup Guide.

Using JDBC Data Sources

Data source operations outlined in *Chapter 6: Database Interaction Fundamentals*, *Chapter 7: Searching and Displaying Data*, and *Chapter 8: Adding and Updating Records* are supported with JDBC data sources. Because JDBC is a standardized API for connecting to tabular data sources, there are few unique tags in LDML 7 that are specific to JDBC data sources or invoke special functions specific to any JDBC data source. The only JDBC-specific LDML tags are for JDBC data sources that support schema ownership (e.g. Frontbase, Sybase), and are described in the *JDBC Schema Tags* section of this chapter.

All LDML tags documented as unique to MySQL data sources in *Chapter 9: MySQL Data Sources* or FileMaker Pro data sources in *Chapter 10: FileMaker Pro Data Sources* are not supported for use with JDBC data sources.

Certification Note: Blue World Communications has tested and certified Microsoft SQL Server 2000 with Microsoft SQL Server 2000 Driver for JDBC for use with Lasso Professional 7 via JDBC. Other JDBC-compliant data sources may be used with Lasso Professional 7, but all features cannot be guaranteed to work by Blue World Communications. See <http://support.blueworld.com> for Support Central articles on connectivity with selected data sources.

Tips for Using JDBC Data Sources

The following is a list of tips to following when writing LDML for use with JDBC data sources. These tips illustrate specific concepts and behaviors to keep in mind when coding, and these tips are most similar to those for MySQL data sources (as opposed to FileMaker Pro data sources).

- Always specify a primary key field using the `-KeyField` command tag in `-Search`, `-Add`, and `-FindAll` actions. This will ensure that the `[KeyField_Value]` tag will always return a value.
- Use `-KeyField` and `-KeyValue` to reference a particular record for updates, duplicates, or deletes.
- Fields may truncate any data beyond the length they are set up to store. Ensure that all fields in JDBC databases have sufficiently long fields for the values that need to be stored in them.
- Use `-ReturnField` command tags to reduce the number of fields which are returned from a `-Search` action. Returning only the fields that need to be used for further processing or shown to the site visitor reduces the

amount of data that needs to travel between Lasso Service and the JDBC data source.

- When an -Add or -Update action is performed on a JDBC database, the data from the added or updated record is returned inside the [Inline] ... [Inline] tags or alternately to the Classic Lasso response page. If the -ReturnField parameter is used, then only those fields specified should be returned from an -Add or -Update action. Setting -MaxRecords=0 can be used as an indication that no record should be returned.
- The -SQL command tag can be allowed or disallowed at the host level for users in Lasso Administration. Once the -SQL command tag is allowed for a user, that user may access any database within the allowed host inside of a SQL statement. For that reason, only trusted users should be allowed to issue SQL queries using the -SQL command tag. For more information, see *Chapter 8: Setting Up Security* in the Lasso Professional 7 Setup Guide.
- SQL statements which are generated using visitor-defined data should be screened carefully for unwanted commands such as DROP or GRANT. See *Chapter 7: Setting Up Data Sources* of the Lasso Professional 7 Setup Guide for more information.
- Always quote any inputs from site visitors that are incorporated into SQL statements. For example, the following SQL SELECT statement includes quotes around the [Action_Param] value. The quotes are escaped \ so they will be embedded within the string rather than ending the string literal. The semi-colon at the end of the statement is optional unless multiple statements are issued.

```
[Variable: 'SQL_Statement']='SELECT * FROM Contacts.People WHERE ' +
  'First_Name LIKE \' + (Action_Param: 'First_Name') + '\';]
```

If [Action_Param] returns John for First_Name then the SQL statement generated by this code would appear as follows.

```
SELECT * FROM Contacts.People WHERE First_Name LIKE 'John';
```

- Lasso Professional 7 uses connection pooling when connecting to data sources via JDBC, and the JDBC connections will remain open during the time that Lasso Professional 7 is running.
- Check for Blue World Support Central articles at <http://support.blueworld.com> for documented issues with using specific JDBC data sources.

JDBC Schema Tags

LDML 7 includes tags that return the user schemas available in a JDBC data source host for the current Lasso Service connection. These tags can only be

used with data sources that use named schema ownership (e.g. Frontbase, Sybase), and complement the other LDML schema and database tags described in *Chapter 6: Database Interaction Fundamentals*.

Note: For information on whether or not your JDBC data source supports named schema ownership, refer to the data source documentation.

Table 1: JDBC Schema Tags

Tag	Description
-Schema	Allows a schema name to be passed as part of an [Inline] ... [/Inline] data source action. The schema name passed here overrides the default schema set for the JDBC data source host in Lasso Administration.
[Schema_Name]	Returns the name of the current schema in use in an [Inline] ... [/Inline] data source action.
[Database_SchemaNames]	Repeats for every schema name in a JDBC data source host available to Lasso. Requires the name of a database in the JDBC data source host as a parameter.
[Database_SchemaNameItem]	Returns the name of the current schema name when used inside [Database_SchemaNames] ... [/Database_SchemaNames] tags.

To reference a schema name in an inline database action:

Use the -Schema command tag to pass the name of the data source schema that should be used for the database action.

```
[Inline: -Show, -Schema='SchemaName', -Database='DBName', -Table='TBName']  
  [Schema_Name]  
[/Inline]
```

→ SchemaName

To list all schema names in a JDBC data source:

Use the [Database_SchemaNames] ... [/Database_SchemaNames] tags to list all databases available in a JDBC data source host. The [Database_SchemaNameItem] tag returns the value of each schema name.

```
[Database_SchemaNames:'DBName']  
  [Database_SchemaNameItem]  
[/Database_SchemaNames]
```

→ SchemaName
SchemaName2



Section III

Programming

This section documents the symbols, tags, expressions, and data types which allow programming logic to be specified within LDML format files. This section contains the following chapters.

- **Chapter 12: *Programming Fundamentals*** introduces basic concepts of LDML programming such as how to output results, how to store and retrieve variables, and how to interact with HTML forms and URLs.
- **Chapter 13: *Conditional Logic*** introduces the [If], [Loop], and [While] tags and demonstrates how they can be used for flow control.
- **Chapter 14: *String Operations*** introduces the string data type and the symbols and tags that can be used to manipulate strings.
- **Chapter 15: *Math Operations*** introduces the integer and decimal data types and the symbols and tags that can be used to perform mathematical operations.
- **Chapter 16: *Date and Time Operations*** introduces the Lasso date format and the tags that can be used to manipulate dates and times.
- **Chapter 17: *Arrays and Maps*** introduces the array, map, and pair data types and the tags that can be used to store and manipulate complex data types.
- **Chapter 18: *Encoding*** explains how strings are encoded in Lasso for output to many different languages and the tags and keywords that can be used to control that output.
- **Chapter 19: *Sessions*** explains how to create server-side variables that maintain their value from page to page while a visitor traverses a Web site.
- **Chapter 20: *Files and Logging*** explains how to log information to files and how to use the file tags to create, read, and write text files.

- *Chapter 21: Error Control* introduces Lasso’s error reporting mechanism and explains how custom error tags can be created and what tags can be used to handle errors which occur while processing a format file.
- *Chapter 22: Control Tags* introduces scheduling, the [Process] tag, page variables, and Lasso administration and security tags.
- *Chapter 23: Miscellaneous Tags* includes documentation of tags that do not fit in any other chapter.
- *Chapter 24: LassoScript* fully documents the alternate script-based syntax for Lasso.

12

Chapter 12

Programming Fundamentals

This chapter introduces the basic concepts of programming using LDML. It is important to understand these concepts before reading the chapters that follow.

- **Overview** explains how to use pages written in LDML and how to deal with errors.
- **Logic vs. Presentation** describes strategies for coding blocks of programming logic code.
- **Data Output** describes strategies for outputting calculation results in HTML or XML.
- **Variables** explains the theory behind variables and how to store and retrieve values.
- **Data Types** explains how to recognize different data types, how to cast between data types, and casting rules.
- **Symbols** is an introduction to symbols and expressions including rules for grouping, precedence, and auto casting.
- **Member Tags** explains how to call member tags and how they differ from process and substitution tags.
- **Forms and URLs** explains how to pass data between pages using HTML forms and URLs and introduces form parameters and tokens.

Overview

LDML is a tag-based scripting language that has all the features of an advanced programming language. LDML has support for data types, object-oriented member tags, mathematical symbols, string symbols, complex nested expressions, logical flow control, threads, and custom tags which can extend Lasso's built-in functions and procedures.

Using Format Files

Format files which contain LDML must be processed by Lasso in order for the embedded tags to be interpreted. The Open... command in a Web browser should not be used to view Lasso format files. Instead, format files should be uploaded to a Web server and loaded with an appropriate URL. For example, a file named `default.lasso` in the root of the Web serving folder might be loaded using the following URL.

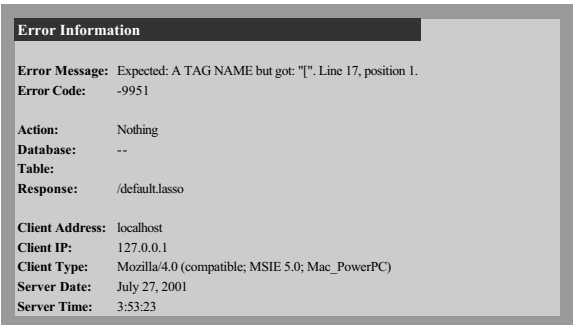
`http://www.example.com/default.lasso`

Simple sequences of tags and LassoScripts can be placed in a text file and then called through the Web browser in order to test LDML programming concepts without the overhead of HTML formatting tags.

Reporting Errors

If there are any LDML syntax errors in a format file which is processed by Lasso, then all processing will stop and an error message will be displayed. The error message will provide the location of the error (if possible) and a description of what syntax caused the error. All errors must be corrected before the page can be fully processed.

Figure 1: Error Page



Note: All valid LDML code above the syntax error will be processed each time the page is loaded. If database actions are being performed, they may be performed each time a page is loaded as long as they are above the point in the page where the error occurs.

Logic vs. Presentation

LDML code can be structured in many ways in order to adapt itself to different coding styles. Some methods involve the tight integration of programming logic (LDML) with page presentation (HTML, XML, and graphics). Other methods involve abstracting the programming logic from the page presentation. LDML offers maximum flexibility for you to determine how you want to structure your pages.

It is often desirable to separate programming logic from page presentation so that different people can work on different aspects of a Web site. For example, an LDML developer can concentrate on creating LassoScripts and blocks of LDML code which define the programming logic of a site. Meanwhile, a Web designer can concentrate on the visual aspects of the Web site with only minimal knowledge of how to integrate LDML into the page presentation so that data is inserted and formatted correctly.

It is also at times desirable for all of your programming to fit tightly within the page presentation. Because LDML is an HTML-like tag language, it is easy to embed LDML within HTML, in effect enhancing static HTML to become dynamic HTML.

The following examples show how to use LDML within HTML as well as how to use LDML abstracted from HTML.

Examples of LDML embedded in HTML:

- LDML tags can be used within HTML markup to insert data from databases, the results of calculations, or LDML commands into otherwise static HTML. The following example inserts the LDML [Image_URL] tag into an HTML tag in order to auto-generate a URL to an image stored in a database.

```

```

- Container tags can be used to hide or show portions of a page. The following example hides an HTML <h2> header unless the variable ShowTitle equals True.

```
[If: (Variable: 'ShowTitle') == True]
<h2>Page Title</h2>
[/If]
```

- Container tags can be used to repeat a portion of a page to present data from many database records or to construct complex HTML tables. The following example shows the fields `First_Name` and `Last_Name` from a database search each in their own row of a constructed table. See *Chapter 6: Database Interaction Fundamentals* for more information about `[Inline]` ... `[/Inline]` tags.

```
[Inline: -Database='Contacts', -Table='People', -KeyField='ID', -FindAll]
<table>
  [Records]
  <tr>
    <td>[Field: 'First_Name'] [Field:'Last_Name']</td>
  </tr>
[/Records]
</table>
[/Inline]
```

Examples of LDML abstracted from HTML:

- LassoScripts can be used to collect programming logic into a block at the top of a format file. Code in the LassoScript can be formatted and commented separate from the HTML in a format file. Separating the programming logic from the page presentation tags allows for easier debugging and customization of format files. The following example shows an `[Inline]` specified in a LassoScript with an `-InlineName` keyword set so the results can be retrieved in the presentation portion of the format file. See *Chapter 24: LassoScript* for more information.

```
<?LassoScript
// This inline finds all records in Contacts.
// The results are fetched using [Records: -InlineName='Results'] ... [/Records]
Inline: -InlineName='Results', -Database='Contacts',-Table='People',-FindAll;
[/Inline;
?>
```

- The `[Include]` tag can be used to include format files that contain portions of the final output. In the following example, the format file shown consists of the standard HTML tags with a pair of `[Include]` tags that insert all of the programming logic from a file named `library.lasso` and the data presentation code from a file named `presentation.lasso`. See *Chapter 20: Files and Logging* for more information about using `[Include]` tags.

```
<html>
<head>
  <title>Lasso FormatFile</title>
  [Include: 'library.lasso']
</head>
```

```

<body>
  [Include: 'presentation.lasso']
</body>
</html>

```

Data Output

The final output of most Lasso format files is an HTML page, XML page, or WML page which will be viewed by a Web site visitor in a client browser. This section describes how the results of expressions can be output and how the output of substitution tags can be controlled.

See also *Chapter 18: Encoding* for more information about using encoding keywords.

Table 1: Output Tags

Tag	Description
[Output]	Outputs the result of a calculation or sub-tag.
[Output_None]	Hides a portion of page from being output, but processes the LDML tags within.
[HTML_Comment]	Surrounds a portion of a page with HTML comment markers, but processes the LDML tags within.

Outputting Values

Substitution tags output values to the format file which is currently being processed in place. Their values are output whether they are contained within LassoScripts or appear intermixed with HTML tags.

The [Output] tag is a general purpose substitution tag which can be used to output the value of any LDML expression, member tag, or sub-tag.

Examples of using the [Output] tag:

- The following LassoScript shows the use of the [Output] tag to return the result of a mathematical expression. The same expression could be placed in the LassoScript without the [Output] tag, but use of the tag makes the result of the LassoScript clearer.

```

<?LassoScript
  Output: 1 + 2 * 3;
?>

```

→ 7

- The [Output] tag allows encoding keywords to be used on the results of string expressions. The following LassoScript shows the use of the [Output] tag to return the result of a string expression with the encoding keyword -EncodeNone applied so the HTML tags are displayed properly on the page.

```
<?LassoScript
  Output: '<b>' + 'Bold Text' + '</b>', -EncodeNone;
?>
```

→ Bold Text

- The results of member tags can be returned using the [Output] tag. This can make the syntax clearer and help to distinguish between member tags that will return a result and those that won't. The following example demonstrates returning the length of a string literal.

```
[Output: 'String Literal'->Length]
```

→ 14

- The [Output] tag is recommended, but not technically required when outputting the values of expressions or member tags. Lasso will interpret any expressions contained in square brackets or the <?LassoScript ... ?> tags. The following expression is equivalent to the [Output] tag shown above.

```
['String Literal'->Length]
```

→ 14

Values which are output without using the [Output] tag do not have any encoding applied by default. The following expression is equivalent to the LassoScript above, but does not require the -EncodeNone encoding keyword be explicitly specified.

```
<?LassoScript
  '<b>' + 'Bold Text' + '</b>';
?>
```

→ Bold Text

Suppressing Output

Sometimes it is desirable to have LDML tags processed in a format file, but not to show the results in the page which is returned to the Web site visitor. The [Output_None] ... [/Output_None] tag can be used to accomplish this purpose. Any LDML tags contained within the container tag will be processed, but the results will not be returned to the Web site visitor.

The following examples use page specific variables in a block of code that will not be output to the user.

```
[Output_None]
  This text will not be returned to the site visitor.
  However, the following tags will be processed.
  [Variable: 'Page Title'='Lasso Format File']
  [Variable: 'Page Error'='None']
[/Output_None]
```

This same example could be written as a LassoScript as follows. The LassoScript will return no value to the page on which it is placed, but any tags within the LassoScript will be processed.

```
<?LassoScript
  Output_None;
  // This LassoScript will return no value.
  // However, the following tags will be processed.
  Variable: 'Page Title'='Lasso Format File';
  Variable: 'Page Error'='None';
[/Output_None;
?>
```

Another way to suppress output is to surround a portion of a page in [HTML_Comment] ... [/HTML_Comment] tags. These tags will become an HTML comment container <!-- ... --> when the page is processed. Any results of the tags inside the container tags will not be shown to the Web site visitor, but will be available if they view the source of the page. This can be useful for providing debugging information which won't affect the overall layout of a Web page. In the following example, the values of several variables are shown in an HTML comment.

```
[HTML_Comment]
  This text will be available in the source of the completed Web page.
  Page Title: [Variable: 'Page Title']
  Page Error: [Variable: 'Page Error']
[/HTML_Comment]

<!--
  This text will be available in the source of the completed Web page.
  Page Title: Lasso Format File
  Page Error: None
?>
```

Variables

Variables are named locations where values can be stored and later retrieved. The concepts of setting and retrieving variables and performing

calculations on variables are essential to understanding how to work with LDML's data types and tags.

Table 2: Variable Tags

Tag	Description
[Variable]	Creates or sets named variables or returns their values.
[Variable_Defined]	Returns True if a variable is defined.
[Var]	Abbreviation of [Variable].
[Var_Defined]	Abbreviation of [Variable_Defined].
[Var_Remove]	Deletes the named variable.

Table 3: Variable Symbols

Symbol	Description
\$	Returns the value of a variable.
#	Returns the value of a local variable.
=	Assigns a value to a variable: \$Variable='NewValue'.

A variable is created and set using the [Variable] tag. The following tag sets a variable named VariableName to the literal string value VariableValue.

```
[Variable: 'VariableName'='VariableValue']
```

A variable is also retrieved using the [Variable] tag. This time, the tag is simply passed the name of the variable to be retrieved. The following tag retrieves the variable named VariableName returning the literal string value VariableValue.

```
[Variable: 'VariableName'] → VariableValue
```

The following LassoScript sets a variable and then retrieves the value. The result of the LassoScript is the value VariableValue.

```
<?LassoScript
  Variable: 'VariableName'='VariableValue';
  Variable: 'VariableName';
?>
```

→ VariableValue

Creating Variables

There is only one way to create a variable, using the [Variable] tag with a name/value parameter. All variables should be created and set to a default value before they are used.

Examples of creating variables:

- An empty variable can be created by setting the variable to ".
[Variable: 'VariableName'='']
- A variable can be created and set to the value of a string literal.
[Variable: 'VariableName'='String Literal']
- A variable can be created and set to the value of an integer or decimal literal.
[Variable: 'VariableName'=123.456]
- A variable can be created and set to the value of any substitution tag such as a field value.
[Variable: 'VariableName'=(Field: 'Field_Name')]

Multiple variables can be created in a single [Variable] tag by listing the name/value parameters defining the variables separated by commas. The following tag defines three variables named x, y, and z.

```
[Variable: 'x'=100, 'y'=324, 'z'=1098]
```

Variable names can be any string literal and case is unimportant. For best results, variables names should start with an alphabetic character, should not contain any punctuation except for underscores and should not contain any white space except for spaces (no returns or tabs). Variable names should be descriptive of what value the variable is expected to contain.

Note: Variables cannot have their value retrieved in the same [Variable] tag they are defined. [Variable: 'x'=10, 'y'=(variable:'x')] is not valid.

Returning Variable Values

The most recent value of a variable can be returned using the [Variable] tag. For example, the following LassoScript creates a variable named VariableName, then retrieves the value of the variable using the [Variable] tag. The result is Variable Value.

```
<?LassoScript
  Variable: 'VariableName'='Variable Value';
  Variable: 'VariableName';
?>
```

→ Variable Value

Variable values can also be retrieved using the \$ symbol. The following LassoScript creates a variable named `VariableName`, then retrieves the value of the variable using the \$ symbol. The result is `Variable Value`.

```
<?LassoScript
  Variable: 'VariableName'='Variable Value';
  Output: $VariableName;
?>
```

→ Variable Value

Setting Variables

Once a variable has been created, it can be set to different values as many times as is needed. The easiest way to set a variable is to use the `[Variable]` tag again just as it was used when the variable was created.

```
[Variable: 'VariableName'='New Value']
```

Variables can also be set using the expression `$VariableName='NewValue'`. This expression should only be used within LassoScripts so that it is not confused with a name/value parameter. This expression can be used to set a variable, but cannot be used to create a variable.

The following LassoScript creates a variable named `VariableName`, sets it to a value `New Value` using an expression, then retrieves the value of the variable. The result is `New Value`.

```
<?LassoScript
  Variable: 'VariableName'='';
  $VariableName='New Value';
  $VariableName;
?>
```

→ New Value

Checking to See if a Variable has been Created

The `[Variable_Defined]` tag can be used to check if a variable has been created and used in the current format file. The following example will return false the first time `[Variable_Defined]` is called, then set the variable using `[Variable]` and return True the second time `[Variable_Defined]` is called.

```
<?LassoScript
  Variable_Defined: 'VariableName';
  Variable: 'VariableName'='VariableValue';
  Variable_Defined: 'VariableName';
?>
```

→ False True

The `[Variable_Defined]` tag will return `True` even if a variable is set to the empty string " (two single quotes with no space) or to `Null`. There is no way to delete a variable once it has been created.

Data Types

Every value in Lasso is defined as belonging to a specific data type. Every value stored in a variable belongs to a specific data type. The data type determines what symbols and member tags are available for use with the value.

Table 4: Data Type Tags

Tag	Description
<code>[Null->Type]</code>	Returns the data type of a value.
<code>[String]</code>	Casts a value to data type string.
<code>[Integer]</code>	Casts a value to data type integer.
<code>[Decimal]</code>	Casts a value to data type decimal.
<code>[Boolean]</code>	Casts a value to data type boolean.
<code>[Date]</code>	Casts a value to data type date.
<code>[Duration]</code>	Casts a value to data type duration.
<code>[Array]</code>	Creates an array data type.
<code>[Map]</code>	Creates a map data type.
<code>[Pair]</code>	Creates a pair data type.
<code>[Bytes]</code>	Creates a bytes data type.

Several data types have already been introduced:

- Strings are sequences of alphanumeric characters. String literals are delimited by single quotes as in 'String Literal'.
- Integers are whole numbers. Integer literals are specified without quotes as in 123 or -987.
- Decimals are numbers which contain a decimal point. Decimal literals are specified without quotes as in 3.1415926 or 24.99.
- Dates are alphanumeric strings that represent a date and/or time. A date must always be cast using the `[Date]` tag in a recognized format to be used as a date data type (e.g. `[Date:'9/29/2002']`).
- Durations are alphanumeric strings that represent a length time (not a 24-hour clock time). A duration must always be cast using the `[Duration]` tag in a recognized format to be used as a duration data type (e.g. `[Duration:'168:00:00']`).

Variables which are set to literal values of a specific data type are themselves said to be of that data type. Variables containing strings are string variables. Any symbols which operate on literal strings will also operate on string variables.

It is important to keep track of what type of value is stored in each variable so that the values of expressions and member tags can be safely predicted.

Returning the Type of a Variable

The [Null->Type] member tag can be used to return the type of a variable or other value. [Null->Type] is a member tag of the data type null which is a precursor to all other data types. The [Null->...] member tags can be used with values of any data type.

The following example shows the value of [Null->Type] for literals of different data types.

```
'String Value'>Type → string
123>Type → integer
9.999>Type → decimal
```

The following example shows the value of [Null->Type] when it is used on a variable which has been set to a string literal.

```
<?LassoScript
  Variable: 'Value' = 'String Value';
  Output: $Value->Type;
?>
```

→ string

The [Null->Type] member tag also works on the compound data types: array, map, and pair. The following example shows the value of [Null->Type] when it is used on a variable which has been set to an array literal.

```
<?LassoScript
  Variable: 'Value' = (Array: 'One', 'Two', 'Three', 'Four');
  Output: $Value->Type;
?>
```

→ array

Casting a Value to a Data Type

Values can be cast from one data type to another in order to ensure that the proper member tags will be available and symbols will work as expected. Each data type defines a tag which has the same name as the data type that can be used to cast a value to that data type.

To cast a value to the string data type:

- Integer and decimal values can be cast to type string using the [String] tag. The value of the string is the same as the value of the integer or decimal value when it is output using the [Variable] tag.

[String: 999.999] → '999.999'

- Boolean values can be cast to type string using the [String] tag. The value will always either be True or False.

[String: True] → 'True'

- Arrays, maps, and pairs should not be cast to type string. The value which results is intended for debugging purposes. More information can be found in *Chapter 17: Arrays and Maps*.

To cast a value to the integer data type:

- Decimal values can be cast to type integer using the [Integer] tag. The value of the decimal number will be truncated at the decimal point. For example, casting 999.999 to type integer results in 999 not 1000.

[Integer: 999.999] → 999

- String values can be cast to type integer using the [Integer] tag. The string must start with a numeric value. For example casting 2String1 to an integer results in 2.

[Integer: '2001: A Space Odyssey'] → 2001

[Integer: '2String1'] → 2

- Boolean values can be cast to type integer using the [Integer] tag. The value of the result will be 1 if the boolean was True or 0 if the boolean was False.

[Integer: True] → 1

[Integer: False] → 0

- Arrays, maps, and pairs should not be cast to type integer. The value which results will always be 0.

To cast a value to the decimal data type:

- Integer values can be cast to type decimal using the [Decimal] tag. The value of the integer number will simply have a decimal point added. For example, casting 123 to type integer results in 123.000000.

[Decimal: 123] → 123.000000

- String values can be cast to type decimal using the [Decimal] tag. The string must start with a numeric value. For example casting 2.5String1 to a decimal results in 2.500000. The 1 at the end of the string is ignored.

[Decimal: '2001: A Space Odyssey'] → 2001.000000

[Decimal: '2.5String1'] → 2.500000

- Boolean values can be cast to type decimal using the [Decimal] tag. The value of the result will be 1.000000 if the boolean was True or 0.000000 if the boolean was False.

[Decimal: True] → 1.000000

[Decimal: False] → 0.000000

- Arrays, maps, and pairs should not be cast to type integer. The value which results will always be 0.000000.

To cast a value to the boolean data type:

- Integer and decimal values can be cast to type boolean using the [Boolean] tag. The value of the boolean will be False if the number is zero or True if the number is non-zero.

[Boolean: 123] → True

[Boolean: 0.0] → False

- String values can be cast to type boolean using the [Boolean] tag. The value of the boolean will be False if the string contains just the word false or is empty and True otherwise.

[Boolean: 'false'] → False

[Boolean: ''] → False

[Boolean: 'true'] → True

[Boolean: 'value'] → True

- Arrays, maps, and pairs should not be cast to type boolean. The value which results will always be False.

To cast a value to the date data type:

- Specially formatted strings may be cast as date data types using the [Date] tag. For a list of date string formats that are automatically recognized as dates, see *Chapter 16: Date and Time Operations*.

[Date: '9/29/2002'] → 9/29/2002 00:00:00

[Date: '9/29/2002 12:30:00'] → 9/29/2002 12:30:00

[Date: '2002-09-29 12:30:00'] → 2002-09-29 12:30:00

- Unrecognized date strings can be cast as date data types using the [Date] tag with the -Format parameter. All eligible date strings must contain numbers, punctuation, and/or allowed words (e.g. February, GMT) in a

format that represents a valid date. For a description of how to format a date string, see *Chapter 16: Date and Time Operations*.

[Date: '9.29.2002', -Format='%m.%d.%Y'] → 9.29.2002

[Date: '20020929', -Format='%Y%m%d'] → 20020929

[Date: 'September 29, 2002', -Format='%B %d, %Y'] → September 29, 2002

To cast a value to the duration data type:

- Specially formatted strings as either hours:minutes:seconds or just seconds may be cast as duration data types using the [Duration] tag. The [Duration] tag always returns values in hours:minutes:seconds format. For more information, see *Chapter 16: Date and Time Operations*.

[Duration: '169:00:00'] → 169:00:00

[Duration: '00:30:00'] → 00:30:00

[Duration: '300'] → 00:05:00

To cast a value to type array, map, or pair:

Values cannot be cast to type array, map, or pair. However, an array, map, or pair can be constructed with the simple data type as its initial value. See *Chapter 17: Arrays and Maps* for more information about how to construct these complex data types.

To cast a value to the bytes data type:

For discussion on the bytes data type, see *Chapter 5: Advanced Programming Topics* in the Extending Lasso 7 Guide.

Automatic Casting

Lasso will cast values to a specific data type automatically when they are used in expressions or as parameters for tags which require a particular type of value. Values will be automatically cast in the following situations:

- Values of every data type are cast to string values when they are output to the Web browser.
- Integer values are cast to decimal values when they are used as parameters in expressions with one integer parameter and one decimal parameter.
- Integer and decimal values are cast to string values when they are used as parameters in expressions with one integer or decimal parameter and one string parameter.
- Values of every data type are cast to boolean values when they are used in logical expressions.

- The [Math_...] tags will automatically cast all parameters to integer or decimal values.
- The [String_...] tags will automatically cast all parameters to string values.

Symbols

Symbols allow for powerful calculations to be performed within LDML tags. The symbols which can be used in expressions are discussed in full detail in the chapter devoted to each data type. String expressions and symbols are discussed in *Chapter 14: String Operations* and decimal and integer expressions and symbols are discussed in *Chapter 15: Math Operations*.

Using Symbols

Since symbols only function on values of a specific data type, values need to be cast to that data type explicitly or they will be automatically cast. For best results, explicit casting should be performed so the meaning of the symbols will be clear. Note that spaces should always be specified between a symbol and its parameters.

As explained in the *Automatic Casting* section above, values used as a parameter in an expression will be automatically cast to a string value if any parameter in the expression is a string value. Integer values will be automatically cast to decimal values. Any value used in a logical expression will be automatically cast to a boolean value.

- The following expression returns 1212 since the integer 12 is automatically cast to a string because one parameter is a string.
[Output: '12' + 12] → 1212
- Similarly, the following expression returns 1212 since the integer 12 is automatically cast to a string because one parameter is a string.
[Output: 12 + '12'] → 1212
- The following expression returns 24 since the string 12 is explicitly cast to an integer.
[Output: (Integer: '12') + 12] → 24
- The following expression returns 24.000000 since the integer 12 is automatically cast to a decimal value because one parameter is a decimal value.
[Output: 12 + 12.0] → 24.000000

- The following expression returns True since the integer 12 is automatically cast to a boolean value True because it is used in a logical expression.

[Output: 12 && 12] → True

When in doubt, the [String], [Integer], and [Decimal] tags should be used to explicitly cast values so that the proper symbols are used.

Note: Always place spaces between a symbol and its parameters. The - symbol can be mistaken for the start of a command tag, keyword, or keyword/value parameter if it is placed adjacent to the parameter that follows.

Assignment Symbols

Variables can be set to the result of an expression, storing that result for later use. For example, the following variable is set to the result of a simple math expression.

[Variable: 'MathResult'=(1 + 2)]

Variables can also be set using assignment symbols within LassoScripts. The equal sign = is the simplest assignment symbol. Other assignment symbols can be formed by combining a decimal, integer, or string symbol with the equal sign. For example, += is the additive assignment symbol.

The following LassoScript creates a variable named MathResult, performs a mathematical operation (adding 4) on it using the additive assignment symbol, and returns the final value.

```
<?LassoScript
  Variable: 'MathResult'=0;
  $MathResult += 4;
  Output: $MathResult;
?>
```

→ 4

The assignment symbol replaces the value of the variable and does not return any output. The assignment expression \$MathResult += 4; is equivalent to the expression \$MathResult = \$MathResult + 4;. Since assignment expressions do not return a value they should only be used within LassoScripts to modify variables.

LassoScripts can use variable results to build very complex operations. For example, the following LassoScript uses several variables to perform a math expression.

```

<?LassoScript
  Variable: 'x'=100, 'y'=4;
  $x = $x / $y;
  $y = $x + $y;
  Output: 'x=' + $x + ' y=' + $y;
?>

```

→ x=25 y=29

Note: If a negative number is used as the right-hand parameter of an assignment symbol it should be surrounded by parentheses.

Member Tags

Member tags are associated with a particular data type and can be used on any value of that data type. The data type of a member tag is represented in the documentation in the member tag name before the member tag symbol `->`. For example, the tag `[String->Length]` can be used with values of data type string, and the tag `[Decimal->SetFormat]` can be used with values of data type decimal.

Member tags are available for string, decimal, integer, date, array, map, and pair data types, and are discussed in detail in *Chapter 14: String Operations*, *Chapter 15: Math Operations*, *Chapter 16: Date and Time Operations*, and *Chapter 17: Arrays and Maps*.

Using Member Tags

Since member tags only function on values of a specific data type, values need to be cast to that data type explicitly. Member tags will not automatically cast values.

For example, the member tag `[String->Length]` can be used to return the length of a string value. If `[String->Length]` is used on a number as in `[Output: 123->Length]` then an error will result:

"Length" was not a member of type "integer"

Instead, the integer must be cast to a string value explicitly before the member tag can be used. The following example returns the length of the string representing the integer correctly.

`[Output: (String: 123)->Length]` → 3

When in doubt, the `[String]`, `[Integer]`, `[Decimal]`, and `[Date]` tags should be used to explicitly cast values so that the proper member tags are available.

Member Tag Types

Member tags can function like either substitution tags which return a value or like process tags which modify the value which the member tag is called on, but do not return a value.

For example, the member tag `[String->Length]` functions like a substitution tag and returns the length of the string on which it is called. The following LassoScript stores a string in a variable `StringVariable` then retrieves its length. The string stored in the variable is left unchanged.

```
<?LassoScript
  Variable: 'StringVariable' = 'A string value';
  Output: $StringVariable->Length;
?>
```

→ 14

In contrast, the member tag `[Decimal->SetFormat]` functions like a process tag, altering the way that a decimal variable will be output when it is cast to a string. The following LassoScript shows the normal decimal value output of a variable.

```
<?LassoScript
  Variable: 'DecimalVariable' = 123.456;
  Output: $DecimalVariable;
?>
```

→ 123.456000

The following LassoScript shows how the output of the decimal value changes when a `[Decimal->SetFormat]` tag is used on the variable `DecimalVariable` to truncate its output to two significant digits.

```
<?LassoScript
  Variable: 'DecimalVariable' = 123.456;
  $DecimalVariable->(SetFormat: -Precision=2);
  Output: $DecimalVariable;
?>
```

→ 123.45

The value stored in the variable `DecimalVariable` is not changed, but the value which is output is formatted according to the rules set in the `[Decimal->SetFormat]` tag.

Forms and URLs

This section discusses how to pass information from format file to format file through HTML forms and URLs. Data can also be passed from format

file to format file using database actions or sessions. Please see *Chapter 6: Database Interaction Fundamentals* and *Chapter 19: Sessions* for more information.

Form Parameters

HTML forms can be used to pass values to an LDML format file. The values are retrieved in the format file using the [Action_Param] tag. Any <input>, <select>, or <textarea> values can be retrieved by name using the [Action_Param] tag except for those which contain LDML command tags.

For example, the following form has two inputs for First_Name and Last_Name and a button that submits the form.

```
<form action="response.lasso" method="POST">
  <p>First Name: <input type="text" name="First_Name" value="">
  <p>Last Name: <input type="text" name="Last_Name" value="">
  <p><input type="submit" name="Submit" value="Submit Value">
</form>
```

In the format file response.lasso—which is loaded when this form is submitted—the following LDML tags will retrieve the values submitted by the site visitor in the form.

```
First Name: [Action_Param: 'First_Name']
Last Name: [Action_Param: 'Last_Name']
```

Even the value of the submit button can be fetched. This can help distinguish between multiple buttons that each have the same name displayed in the Web browser.

```
Button Value: [Action_Param: 'Submit']
```

URL Parameters

URLs can be used to pass values to an LDML format file. The values are retrieved in the format file using the [Action_Param] tag. Any values which are passed as URL parameters can be retrieved by name using the [Action_Param] tag except for those which contain LDML command tags.

For example, the URL in the following anchor tag has two parameters for First_Name and Last_Name.

```
<a href="response.lasso?First_Name=John&Last_Name=Doe">John Doe</a>
```

In the format file response.lasso—which is loaded when this form is submitted—the following LDML tags will retrieve the values submitted by the site visitor on the form.

```
First Name: [Action_Param: 'First_Name']
Last Name: [Action_Param: 'Last_Name']
```

13

Chapter 13

Conditional Logic

Conditional tags allow programming logic to be embedded into format files. Portions of a page can be hidden or repeated multiple times. Code can be executed in every repetition of a loop or every several repetitions. Complex decision trees can be created which execute code only under very specific conditions.

- *If Else Conditionals* explains how to use the [If] ... [/If] tags and [Else] tag to conditionally determine the results of a page or to execute LDML code.
- *Select Statements* explains how to use [Select] ... [Case] ... [/Select] tags to choose what code to execute based on the value of a variable.
- *Loops* explains how to use the [Loop] ... [/Loop] tags to repeat a portion of the page and documents the [Loop_Abort] and [Loop_Count] tags used in any repeating container tag.
- *Iterations* explains how to use the [Iterate] ... [/Iterate] tags to perform an action using the value of each element of a compound data type in turn.
- *While Loops* explains how to use the [While] ... [/While] tags to repeat a portion of a page while a condition is True.
- *Abort Tag* explains how to use the [Abort] tag to halt execution of a format file.
- *Boolean Data Type* describes the [Boolean] tag and boolean symbols which can be used to create complex conditional expressions.

If Else Conditionals

Code can be conditionally executed and page elements can be conditionally shown by placing them within `[If] ... [/If]` container tags. The code or other page elements will only be processed if the expression in the opening `[If]` tag evaluates to `True`.

```
[If: (Variable: 'Test') == True]
  This text will be shown if the variable Test equals True.
[/If]
```

The `[Else]` tag allows for either/or logic to be programmed. If the condition in the `[If]` tag is `True` then the code between the `[If]` tag and the `[Else]` tag is processed, otherwise the code between the `[Else]` tag and the closing `[/If]` tag is processed.

```
[If: (Variable: 'Test') == True]
  This text will be shown if the variable Test equals True.
[Else]
  This text will be shown if the variable Test does not equal True.
[/If]
```

A series of tests can be made and code associated with the first test that returns `True` can be shown by specifying expressions within the `[Else]` tags. The code between the `[Else]` tag with a conditional expression and the next `[Else]` tag will only be shown if the expression returns `True`. As many `[Else]` tags as needed can be specified within a single set of `[If] ... [/If]` container tags.

Note: The `[Select] ... [Case] ... [/Select]` tags can be used to perform a similar operation. These tags are discussed in the next section.

```
[If: (Variable: 'Test') == (-1)]
  This text will be shown if the variable Test equals -1.
[Else: (Variable: 'Test') == 2]
  This text will be shown if the variable Test equals 2.
[Else: (Variable: 'Test') == 3]
  This text will be shown if the variable Test equals 3.
[/If]
```

A final `[Else]` tag without a conditional expression can be included. The code between the `[Else]` tag and the closing `[/If]` tag will only be processed if the expression in the opening `[If]` tag returns `False` and the expressions in all subsequent `[Else]` tags return `False` as well.

```
[If: (Variable: 'Test') == 1]
  This text will be shown if the variable Test equals 1.
[Else: (Variable: 'Test') == 2]
  This text will be shown if the variable Test equals 2.
[Else: (Variable: 'Test') == 3]
```

```

    This text will be shown if the variable Test equals 3.
[Else]
    This text will be shown if the variable Test is not equal to 1, 2, or 3.
[/If]

```

Table 1: If Else Tags

Tag	Description
[If] ... [/If]	Executes the contents of the container only if the expression in the [If] tag returns True.
[Else]	Valid only within [If] ... [/If] container tags. Executes the remainder of the container tag only if the expression in the [Else] tag returns True or no expression is specified.

The rules for specifying expressions in the [If] and [Else] tags are presented in full in the following section entitled *Boolean Data Type*.

Note: The [If] and [Else] tags will simply output the result of the specified conditional expression parameter if they are called individually on a page, i.e. not as part of a valid [If] ... [Else] ... [/If] container tag.

To conditionally execute code within a LassoScript:

Use the [If] tag with an appropriate conditional expression. In the following example, an [Output] tag will only be processed if the current username returned by the [Client_Username] tag is Anonymous.

```

<?LassoScript
    If: ((Client_Username) == 'Anonymous');
        Output: 'You are an anonymous user';
    /If;
?>

```

To show a different portion of a page if an error occurs:

Errors are reported in Lasso using the [Error_CurrentError] tag. This tag can be compared with many specific error type tags to check to see if a particular error occurred. In the following example, the current error is compared to [Error_SecurityError] in order to display an appropriate message.

```

[If: (Error_CurrentError) == (Error_SecurityError)]
    You don't have permission to access that resource.
[/If]

```

Note: See *Chapter 21: Error Control* for more information about the [Error_...] tags.

Complex Conditionals

There are two methods for creating complex conditionals. Each of these methods can be used interchangeably depending on what conditions need to be checked and the preference of the Lasso developer.

Examples of complex conditionals

- The conditional expression within the opening [If] tag can be used to check several different conditions. The conditions are appended using the and && symbol which returns True if both parameters return True or the or || symbol which returns True if either parameter returns True.

In the following example, two fields from a database are checked to determine what title to put on a salutation. The Sex field is checked to see if the visitor is Male or Female and the Married field is checked to see if the visitor is Married or Single. Compound conditional expressions are created to check for the combination of gender and marriage status for each title.

```
[If: ((Field: 'Sex') == 'Male')]
  Dear Mr. [Field: 'First_Name'] [Field: 'Last_Name'],
[Else: ((Field: 'Sex') == 'Female') && ((Field: 'Marriage') == 'Married')]
  Dear Mrs. [Field: 'First_Name'] [Field: 'Last_Name'],
[Else: ((Field: 'Sex') == 'Female') && ((Field: 'Marriage') == 'Single')]
  Dear Ms. [Field: 'First_Name'] [Field: 'Last_Name'],
[Else]
  To whom it may concern,
[/If]
```

- Nested [If] ... [/If] tags can be used to check several conditions in turn. The conditional expression in each [If] tag is simple, but the nesting establishes that the innermost [If] ... [/If] tags are only executed if the outermost [If] ... [/If] tags evaluate their conditional expression to True.

In the following example the [If] ... [/If] tags cause the Marriage field to be evaluated if the conditional expression in the outermost [Else] tag finds that the Sex field contains Female.

```
[If: ((Field: 'Sex') == 'Male')]
  Dear Mr. [Field: 'First_Name'] [Field: 'Last_Name'],
[Else: ((Field: 'Sex') == 'Female')]
  [If: ((Field: 'Marriage') == 'Married')]
    Dear Mrs. [Field: 'First_Name'] [Field: 'Last_Name'],
  [Else: ((Field: 'Marriage') == 'Single')]
    Dear Ms. [Field: 'First_Name'] [Field: 'Last_Name'],
  [/If]
[/If]
```

Select Statements

Select statements can be used when a variable can take multiple values and a different block of code should be executed depending on the current value. The variable to be checked is specified in the opening `[Select]` tag. A series of `[Case]` tags follow, each specified with a possible value of the variable. If one of the `[Case]` tags matches the value of the variable then the code until the next `[Case]` tag or the closing `[/Select]` tag will be executed.

For example, to return different text depending on value a variable named `Test` current has the following `[Select] ... [/Select]` statement could be used.

```
[Select: (Variable: 'Test')]
[Case (-1)]
    This text will be shown if the variable Test equals -1.
[Case: 2]
    This text will be shown if the variable Test equals 2.
[Case: 3]
    This text will be shown if the variable Test equals 3.
[/Select]
```

A `[Case]` tag without any value is used as the default value for the `[Select] ... [/Select]` statement in the event that no `[Case]` statement matches the value of the parameter of the opening `[Select]` tag. The first `[Case]` tag without any value is returned as the default value.

```
[Select: (Variable: 'Test')]
[Case (-1)]
    This text will be shown if the variable Test equals -1.
[Case: 2]
    This text will be shown if the variable Test equals 2.
[Case: 3]
    This text will be shown if the variable Test equals 3.
[Case]
    This text is shown if the variable does not equal any of the values.
[/Select]
```

Table 2: Select Tags

Tag	Description
<code>[Select] ... [/Select]</code>	Takes a single parameter which is used to decide which enclosed <code>[Case]</code> tag to select. Requires one or more <code>[Case]</code> tags to be specified. Returns the value of the code between the selected <code>[Case]</code> statement and the next <code>[Case]</code> statement or the closing <code>[/Select]</code> tag.
<code>[Case]</code>	Accepts a single parameter which is checked against the parameter of the enclosing <code>[Select]</code> tag. If no parameter is specified then the tag defines the default case.

To return a different value based on the type of a variable:

Use the [Select] ... [Case] ... [/Select] tags to return a different value depending on the type of a variable. The following code outputs the value of a variable named MyVariable that could be of any type. If the variable is not of any built-in type then the default output is to cast it to string.

```
[Select: (Variable: 'MyVariable')->Type]
  [Case: 'Integer']
    <br>Integer value [Variable: 'MyVariable'].
  [Case: 'Decimal']
    <br>Decimal value [Variable: 'MyVariable'].
  [Case: 'String']
    <br>String value [Variable: 'MyVariable'].
  [Case: 'Boolean']
    <br>Boolean value [Variable: 'MyVariable'].
  [Case: 'Array']
    <br>Array value [Variable: 'MyVariable'].
  [Case: 'Map']
    <br>Map value [Variable: 'MyVariable'].
  [Case: 'Pair']
    <br>Pair value [Variable: 'MyVariable'].
  [Case]
    <br>Unknown type value [String: (Variable: 'MyVariable')].
[/Select]
```

Loops

A portion of a page can be repeated a number of times using the [Loop] ... [/Loop] tags. The parameters to the opening [Loop] tag define how many times the portion of the page should be repeated. For example, a message in a Web page could be repeated five times using the following [Loop] tag.

```
[Loop: 5]
  <br>This is repeated five times.
[/Loop]
```

→
This is repeated five times.

This is repeated five times.

This is repeated five times.

This is repeated five times.

This is repeated five times.

The basic form of the [Loop] ... [/Loop] tags simply repeats the contents of the tags as many times as is specified by the parameter. The opening [Loop] tag can also accept a number of keyword/value parameters to create more complex repetitions.

Table 3: [Loop] Tag Parameters

Keyword	Description
-From	Specifies the starting repetition for the [Loop] tag. Can also be specified as -LoopFrom.
-To	Specifies the ending repetition for the [Loop] tag. Can also be specified as -LoopTo.
-By	Specifies how many repetitions should be skipped on each actual repetition of the contents of the [Loop] ... [/Loop] tag. Can also be specified as -LoopIncrement.

The following example shows a loop that runs backward for five repetitions by setting -From to 5, -To to 1 and -By to -1. The [Loop_Count] tag shows the number of the current repetition.

```
[Loop: -From=5, -To=1, -By=-1]
  <br>This is repetition number [Loop_Count].
[/Loop]
```

→
This is repetition number 5.

This is repetition number 4.

This is repetition number 3.

This is repetition number 2.

This is repetition number 1.

Note: The [Loop_Count] tag can be used in any looping container tag within LDML to return the number of the current repetition. This includes the [Records] ... [/Records] tags.

The [Loop_Abort] tag can be used to halt a [Loop] before it reaches the specified number of repetitions. In the following example, the [Loop] tag is stopped after the third repetition by checking to see if [Loop_Count] is equal to 3.

```
[Loop: 5]
  <br>This is repeated five times.
  [If: (Loop_Count) == 3]
    [Loop_Abort]
  [/If]
[/Loop]
```

→
This is repeated five times.

This is repeated five times.

This is repeated five times.

Note: The [Loop_Abort] tag can be used in any looping container tag within LDML to abort the loop. This includes the [Records] ... [/Records] tags.

The modulus symbol % can be used in an [If] ... [/If] conditional to perform a task on every other repetition (or every nth repetition). The conditional expression (Loop_Count % 2)==0 returns True for every other repetition of the loop.

```
[Loop: 5]
  [If: (Loop_Count % 2) == 0]
    <br>This is an Even loop.
  [Else]
    <br>This is an Odd loop.
  [/If]
[/Loop]

→ <br>This is an odd loop.
   <br>This is an even loop.
   <br>This is an odd loop.
   <br>This is an even loop.
   <br>This is an odd loop.
```

The modulus symbol can be used in any looping container tag within LDML to show elements in alternate rows. This includes the [Records] ... [/Records] tags.

Note: The [Repetition] tag from earlier versions of Lasso has been deprecated. It's use is not recommended. Any code using the [Repetition] tag should be changed to the modulus operator for dramatically better speed and future compatibility.

Table 4: Loop Tags

Tag	Description
[Loop] ... [/Loop]	Repeats the contents of the container tag a specified number of times.
[Loop_Count]	Returns the number of the current repetition.
[Loop_Abort]	Aborts the [Loop] ... [/Loop] tag, jumping immediately to the closing tag.

To list all the field names for a table:

An [Inline] ... [/Inline] with a -Show command tag can be used to get a list of all the field names in a table. The [Field_Name] tag accepts a -Count parameter that returns how many fields are in the current table or an integer parameter that returns the name of one of the fields. The following example uses the [Loop] ... [/Loop] tags to display a list of all the field names in a table.

```
[Inline: -Database='Contacts', -Table='People', -Show]
  [Loop: (Field_Name: -Count)]
    <br>[Field_Name: (Loop_Count)]
  [/Loop]
[/Inline]
```

```
→ ID
  First_Name
  Last_Name
```

To loop through the elements of an array:

The elements of an array can be displayed to a site visitor or otherwise manipulated by looping through the array using the [Loop] ... [/Loop] tags. The [Array->Size] tag returns the number of elements in an array and the [Array->Get] tag returns a specific element by index. The following example shows how to store the names of the days of the week in an array and then list those elements using [Loop] ... [/Loop] tags.

```
<?LassoScript
  Encode_Set: -EncodeNone;

  Variable: 'DaysOfWeek' = (Array: 'Sunday', 'Monday', 'Tuesday',
    'Wednesday', 'Thursday', 'Friday', 'Saturday');

  Loop: ($DaysOfWeek->Size);
    Output: '<br>' + $DaysOfWeek->(Get: (Loop_Count));
  /Loop;

  /Encode_Set;
?>
```

```
→ <br>Sunday
  <br>Monday
  <br>Tuesday
  <br>Wednesday
  <br>Thursday
  <br>Friday
  <br>Saturday
```

Note: See *Chapter 17: Arrays and Maps* for more information about the array member tags.

To format a found set in two columns:

The modulus symbol % can be used to format a found set in two columns. In the following example, an HTML <table> is constructed with one cell for each person found by an [Inline] ... [/Inline] based -FindAll action. The modulus symbol % is used to insert the row tags every other record.

```

[Inline: -Database='Contacts', -Table='People', -FindAll]
<table>
  <tr>
    [Records]
    <td>[Field: 'First_Name'] [Field: 'Last_Name']</td>
    [If: (Loop_Count % 2) == 0]
    </tr><tr>
      [If]
    [Records]
    </tr>
  </table>
[/Inline]
→ <table>
  <tr>
    <td>Jane Person</td>
    <td>John Person</td>
  </tr><tr>
    <td>Joe Surname</td>
  </tr>
</table>

```

Iterations

The `[Iterate] ... [/Iterate]` tags loop through each element of a complex data type such as an array or a map. A variable is set to the value of each element of the complex data type in turn. This allows the same operation to be performed on each element.

Note: The `[Iterate] ... [/Iterate]` tags can be used with built-in array, map, pair, and string data types. It can also be used with any custom data type that supports the `[Type->Size]` and `[Type->Get]` member tags.

For example, to print out each element of an array stored in a variable `myArray` the following tags could be used. The opening `[Iterate]` tag contains the name of the variable storing the array and a definition for the variable that should be set to each element of the array in turn. In this case a new variable `myItem` will be created. The value for `myItem` is then output within the `[Iterate] ... [/Iterate]` tags.

```

[Variable: 'myArray' = (Array: 'Winter', 'Spring', 'Summer', 'Autumn')]
[Iterate: (Variable: 'myArray'), (Variable: 'myItem')]
  <br>The season is: [Variable: 'myItem'].
[/Iterate]

```

```
→ <br>The season is: Winter.
    <br>The season is: Spring.
    <br>The season is: Summer.
    <br>The seasons is: Fall.
```

The [Iterate] ... [/Iterate] tags are equivalent to using [Loop] ... [/Loop] tags to cylce through each element of a complex data type, but are significantly easier to use and provide faster operation.

Table 5: Iteration Tags

Tag	Description
[Iterate] ... [/Iterate]	Cycles through each element of a compound data type in turn. The opening tag accepts two parameters. The first is the compound data type to be iterated through. The second is a reference to a variable which should be set to the value of each element of the first parameter in turn.

Note: The second parameter to the opening [Iterate] tag should either be of the form (Variable: 'NewVariableName') or should reference an existing variable using \$ExistingVariable. The \$ symbol cannot be used to create a new variable.

To print out each character of a string:

Use the [Iterate] ... [/Iterate] tags to cycle through each character of the string in turn. The following code prints out each character of a string on a separate line.

```
[Variable: 'myString'='blue']
[Iterate: $myString, (Variable: 'myCharacter')]
    <br>[Variable: 'myCharacter']
[/Iterate]

→ <br>b
    <br>l
    <br>u
    <br>e
```

While Loops

[While] ... [/While] tags allow a portion of a page to repeat while a specified conditional expression returns True. The expression specified in the opening [While] tag is checked on each pass through the loop and if the expression returns True then the contents are displayed again.

In the following example, a variable `ConditionVariable` is set to `True`. Once the `[Loop_Count]` is greater than 3 the variable is set to `False`, ending the `[While] ... [/While]` loop.

```
[Variable: 'ConditionVariable' = True]
[While: ($ConditionVariable == True)]
  <br>This is repetition [Loop_Count]
  [If: (Loop_Count) >= 3]
    [Variable: 'ConditionVariable' = False]
  [/If]
[/Loop]

→ <br>This is repetition 1.
   <br>This is repetition 2.
   <br>This is repetition 3.
```

Table 6: While Tags

Tag	Description
[While] ... [/While]	Repeats the contents of the container tag until the condition specified in the opening tag returns False.
[Loop_Count]	Returns the number of the current repetition.
[Loop_Abort]	Aborts the [While] ... [/While] tag, jumping immediately to the closing tag.

Abort Tag

The `[Abort]` tag can be used to abort the execution of the current format file. This can be useful in a situation where an error has occurred that prevents the rest of the file from executing. An `[Abort]` can be used after a `[Redirect_URL]` so Lasso does not need to process the rest of the page before sending the redirect to the client. Finally, an `[Abort]` can be used in a custom error page in order to prevent the standard error message from being shown at the bottom of the page.

Table 7: Abort Tag

Tag	Description
[Abort]	Aborts the current format file, returning all of the content which has been created so far to the client.

To speed up a [Redirect_URL]:

Use the [Abort] tag immediately after the [Redirect_URL] tag. All LDML code after the [Abort] tag will be ignored so the [Redirect_URL] tag's modifications to the HTTP response will be sent to the client immediately.

```
[Redirect_URL: 'http://www.example.com/']
[Abort]
```

Boolean Type

The boolean data type simply represents True or False. All comparison symbols and boolean symbols in LDML return a value of the boolean data type.

The following values are equivalent to each of the boolean values both when automatically cast and when explicitly cast using the [Boolean] tag. However, it is recommended that you use True and False whenever possible to avoid confusion.

- True is equivalent to any positive integer or decimal such as 1, 45, or 100.15, any non-empty string such as 'String', or any non-null data type such as an array, map, or pair.
- False is equivalent to integer 0 or decimal 0.0, the empty string "", or Null.

Note: The string 'True' happens to be equivalent to True, but the string 'False' is not equivalent to False. Always type the boolean values True and False without quotation marks.

Table 8: Boolean Tag

Tag	Description
[Boolean]	Casts a value to a boolean value.

The boolean data type is most commonly associated with conditional expressions such as those specified in the opening [If] or [While] tags. Any conditional expression which uses a conditional symbol such as ==, !=, <, <=, >, >=, or >> will return a boolean value. Multiple conditional expressions can be combined using any of the boolean symbols detailed in *Table 9: Boolean Symbols*.

Table 9: Boolean Symbols

Symbol	Description
&&	And. Returns True if both parameters are True.
	Or. Returns True if either parameter is True.

!	Not. Returns False if the parameter following is True.
==	Equality. Returns True if both parameters are equal.
!=	Inequality. Returns True if both parameters are different.

Note: Single parameter expressions must be surrounded by parentheses if they are used on the right hand side of a boolean symbol.

To check for two conditions in an [If] tag:

- In order to return True if both conditions are True use the && symbol.

```
[If: ($Condition1 == True) && ($Condition2 == True)]
  Both conditions are True.
[/If]
```
- In order to return True if either of the conditions is True use the || symbol.

```
[If: ($Condition1 == True) || ($Condition2 == True)]
  One of the conditions is True.
[/If]
```
- In order to return True if a condition is False use the ! symbol.

```
[If: !($Condition1 == True)]
  The condition is False.
[/If]
```
- In order to return True if the two conditions are equal (both True or both False) use the == symbol.

```
[If: ($Condition1 == True) == ($Condition2 == True)]
  Both conditions are True or both conditions are False.
[/If]
```
- In order to return True if the two conditions are not equal (one is True and the other is False) use the != symbol.

```
[If: ($Condition1 == True) != ($Condition2 == True)]
  One condition is True and the other is False.
[/If]
```

To use single parameter symbols in a comparison:

If expressions using the single-parameter symbols !, -, and + are going to be used as the second parameter to a comparison symbol, they should be surrounded by parentheses.

- To compare a variable to -1 use parentheses around -1 on the right-hand side of the comparison operator.


```
[If: ($Variable == (-1))  
  The variable is equal to -1.  
[Else: ($Variable > (-1))  
  The variable is greater than -1.  
[Else: ($Variable < (-1))  
  The variable is less than -1.  
[/If]
```

- To compare a variable to the negation of an expression, use parentheses around the entire right-hand side of the comparison operator.

```
[If: ($Variable == (!True))  
  The variable is not equal to False.  
[/If]
```

Note: These expressions can usually be rewritten with the opposite comparison symbol or by using the negation symbol around the entire conditional expression.

14

Chapter 14

String Operations

Text in Lasso is stored and manipulated using the string data type or the [String] tags. This chapter details the symbols and tags that can be used to manipulate string values.

- *Overview* provides an introduction to the string data type and how to cast values to and from other data types.
- *String Symbols* details the symbols that can be used to create string expressions.
- *String Manipulation Tags* describe the member and substitution tags that can be used to modify string values.
- *String Conversion Tags* describes the member and substitution tags that can be used to convert the case of string values.
- *String Validation Tags* describes the member and substitution tags that can be used to compare strings.
- *String Information Tags* describes the member and substitution tags that can be used to get information about strings and characters.
- *String Casting Tags* describes the [String->Split] tag which can be used to cast a string to an array value.
- *Regular Expressions* describes the string tags that allow for regular expression substitutions.

Overview

Many LDML tags are dedicated to outputting and manipulating text. LDML is used to format text-based HTML pages or XML data for output. LDML is also used to process and manipulate text-based HTML form inputs and URLs. Text processing is a central function of LDML.

As a result of this focus on text processing, the string data type is the primary data type in LDML. When necessary, all values are cast to string before subsequent tag or symbol processing occurs. All values are cast to string before they are output into the HTML page or XML data which will be served to the site visitor.

There are three types of operations that can be performed directly on strings.

- Symbols can be used to perform string calculations within LDML tags or to perform assignment operations within LassoScripts.

[Output: 'The' + ' ' + 'String'] → The String

- Member tags can be used to manipulate string values or to output portions of a string.

[Output: 'The String'-(Substring: 4, 6)] → String

- Substitution tags can be used to test the attributes of strings or to modify string values.

[String_LowerCase: 'The String'] → the string

Each of these methods is described in detail in the sections that follow.

This guide contains a description of every symbol and tag and many examples of their use. The LDML Reference is the primary documentation source for LDML symbols and tags. It contains a full description of each symbol and tag including details about each parameter.

Unicode Characters

Lasso Professional 7 supports the processing of Unicode characters in all string tags. The escape sequence `\u...` can be used with 4, or 8 hexadecimal characters to embed a Unicode character in a string. For example `\u002F` represents a `/` character, `\u0020` represents a space, and `\u0042` represents a capital letter B. The same type of escape sequence can be used to embed any Unicode character `\u4E26` represents the Traditional Chinese character 並.

Lasso also supports common escape sequences including `\r` for a return character, `\n` for a new-line character, `\r\n` for a Windows return/new-line, `\f` for a form-feed character, `\t` for a tab, and `\v` for a vertical-tab.

Casting Values to Strings

Values can be cast to the string data type automatically in many situations or they can be cast explicitly using the [String] tag.

Table 1: String Tag

Tag	Description
[String]	Casts a value to type string.

Examples of automatic string casting:

- Integer and decimal values are cast to strings automatically if they are used as a parameter to a string symbol. If either of the parameters to the symbol is a string then the other parameter is cast to a string automatically. The following example shows how the integer 123 is automatically cast to a string because the other parameter of the + symbol is the string String.

[Output: 'String ' + 123] → String 123

The following example shows how a variable that contains the integer 123 is automatically cast to a string.

[Variable: 'Number' = 123]

[Output: 'String ' + (Variable: 'Number')] → String 123

- Array, map, and pair values are cast to strings automatically when they are output to a Web page. The value they return is intended for the developer to be able to see the contents of the complex data type and is not intended to be displayed to site visitors.

[Output: (Array: 'One', 'Two', 'Three')]

→ (Array: (One), (Two), (Three))

[Output: (Map: 'Key1'='Value1', 'Key2'='Value2')]

→ (Map: (Key1)=(Value1), (Key2)=(Value2))

[Output: (Pair: 'Name'='Value')]

→ (Pair: (Name)=(Value))

More information can be found in *Chapter 17: Arrays and Maps*.

- The parameters for string substitution tags are automatically cast to strings. The following example shows how to use the [String_Length] substitution tag on a numeric value from a field.

[Field: 'Age'] → 21

[String_Length: (Field: 'Age')] → 2

To explicitly cast a value to the string data type:

- Integer and decimal values can be cast to type string using the [String] tag. The value of the string is the same as the value of the integer or decimal value when it is output using the [Variable] tag.

The following example shows a math calculation and the integer operation result 579. The next line shows the same calculation with string parameters and the string symbol result 123456.

```
[Output: 123 + 456] → 579
[Output: (String: 123) + (String: 456)] → 123456
```

- Boolean values can be cast to type string using the [String] tag. The value will always either be True or False. The following example shows a conditional result cast to type string.

```
[Output: (String: ('dog' == 'cat'))] → false
```

- String member tags can be used on any value by first casting that value to a string using the [String] tag. The following example shows how to use the [String->Size] member tag on a numeric value from a field by first casting the field value to type string.

```
[Field: 'Age'] → 21
[Output: (String: (Field: 'Age'))->Size] → 2
```

String Symbols

The easiest way to manipulate values of the string data type is to use the string symbols. *Table 2: String Symbols* details all the symbols that can be used with string values.

Table 2: String Symbols

Symbol	Description
+	Concatenates two strings. This symbol should always be separated from its parameters by a space.
-	Deletes a substring. The first occurrence of the right parameter is deleted from the left parameter. This symbol should always be separated from its parameters by a space.
*	Repeats a string. The right parameter should be a number.
=	Assigns the right parameter to the variable designated by the left parameter.

<code>+=</code>	Concatenates the right parameter to the value of the left parameter and assigns the result to the variable designated by the left parameter.
<code>-=</code>	Deletes the right parameter from the value of the left parameter and assigns the result to the variable designated by the left parameter.
<code>*=</code>	Repeats the value of the left parameter and assigns the result to the variable designated by the left parameter.
<code>>></code>	Returns True if the left parameter contains the right parameter as a substring.
<code>!>></code>	Returns True if the left parameter does not contain the right parameter as a substring.
<code>==</code>	Returns True if the parameters are equal.
<code>!=</code>	Returns True if the parameters are not equal.
<code><</code>	Returns True if the left parameter comes before the right parameter alphabetically.
<code><=</code>	Returns True if the left parameter comes before the right parameter alphabetically or if the parameters are equal.
<code>></code>	Returns True if the left parameter comes after the right parameter alphabetically.
<code>>=</code>	Returns True if the left parameter comes after the right parameter alphabetically or if the parameters are equal.
<code>===</code>	Returns True if the parameters are equal and both are of type string. No casting is performed.

Each of the string symbols takes two parameters. One of the parameters must be a string value in order for the symbol to perform the designated string operation. Many of the symbols can also be used to perform integer or decimal operations. If both parameters are integer or decimal values then the mathematical operation defined by the symbol will be performed rather than the string operation.

As long as one of the parameters of the symbol is a string the other parameter will be auto-cast to a string value before the operation defined by the symbol is performed. The two exceptions to this are the `*` and `*=` symbols which must have an integer as the right parameter.

Note: Full documentation and examples for each of the string symbols can be found in the LDML Reference.

Examples of using the string symbols:

- Two strings can be concatenated using the `+` symbol. Note that the symbol is separated from its parameters using spaces.

[Output: 'Alpha ' + 'Beta'] → Alpha Beta

- A string and an integer can be concatenated using the + symbol. The integer will be automatically cast to a string. Note that the symbol is separated from its parameters using spaces.

[Output: 'Alpha ' + 1000] → Alpha 1000

- A substring can be deleted from a string using the - symbol. The following example shows how to remove the substrings and from a string of HTML text. Note that the symbol is separated from its parameters using spaces.

[Output: 'Bold Text' - '' - ''] → Bold Text

- A string can be repeated using the * symbol. The following example shows how to repeat the word Lasso three times.

[Output: 'Lasso ' * 3] → Lasso Lasso Lasso

- Strings will be automatically concatenated even if the + symbol is omitted. This makes concatenating long sets of strings easier.

[Output: 'Alpha ' 'Beta'] → Alpha Beta

Examples of using the string assignment symbols:

- A string variable can be assigned a new value using the = symbol. The following example shows how to define a string symbol and then set it to a new value. The new value is output using the [Output] tag.

```
<?LassoScript
  Variable: 'StringVariable' = 'The String Value';
  $StringVariable = 'New String Value';
  Output: $StringVariable;
?>
```

→ New String Variable

- A string variable can be used as a collector by concatenating new values to it in place using the += symbol. The following example shows how to define a string symbol and then concatenate several values to it. The final value is output using the [Output] tag.

```
<?LassoScript
  Variable: 'StringVariable' = 'The ';
  $StringVariable += 'String ';
  $StringVariable += 'Variable';
  Output: $StringVariable;
?>
```

→ The String Variable

Examples of using the string comparison symbols:

- Two strings can be compared for equality using the == symbol and != symbol. The result is a boolean True or False.
 [Output: 'Alpha' == 'Beta'] → False
 [Output: 'Alpha' != 'Beta'] → True
- Strings can be ordered alphabetically using the <, <=, >, and >= symbols. The result is a boolean True or False.
 [Output: 'Alpha' > 'Beta'] → False
 [Output: 'Alpha' < 'Beta'] → True
- A string can be checked to see if it contains a particular substring using the >> symbol. The result is a boolean True or False.
 [Output: 'Bold Text' >> ''] → True

String Manipulation Tags

The string data type includes many tags that can be used to manipulate string values. The available member tags are listed in *Table 3: String Manipulation Member Tags* and the available substitution tags are listed in *Table 4: String Manipulation Tags*.

In addition to the tags in this section, the tags in the following section on *String Conversion Tags* can be used to modify the case of a string and the tags in the section on *Regular Expression Tags* can be used for more powerful string manipulations using regular expressions.

The member tags in this section all modify the base string in place and do not return a value. For example, the [String->Append] tag works like the += symbol. In order to see the values that were appended to the string, the variable containing the string must be output using the [Output] tag.

```
[Variable: 'myString' = 'Test']
[$myString->(Append: ' string.')]
[Output: $myString] → Test string.
```

In contrast, the substitution tags return the modified string directly.

```
[String_Concatenate: 'Test', ' string.'] → Test string.
```

The member tags should be used when multiple modifications need to be made to a string that is stored in a variable. The substitution tags, or string symbols, can be used when the value is required immediately for output.

Table 3: String Manipulation Member Tags

Tag	Description
[String->Append]	Casts the parameters to strings and appends them to the string. Modifies the string and returns no value. Requires one string parameter.
[String->Merge]	Inserts a merge string into the string. Requires two parameters, the location at which to insert the merge string and the string to insert. Optional third and fourth parameters specify an offset into the merge string and number of characters of the merge string to insert.
[String->PadLeading]	Pads the front of a string to a specified length with a pad character. Modifies the string and returns no value. Requires a length to pad the string. Optional second parameter is the padding character (defaults to space).
[String->PadTrailing]	Pads the end of a string to a specified length with a pad character. Modifies the string and returns no value. Requires a length to pad the string. Optional second parameter is the padding character (defaults to space).
[String->Remove]	Removes a substring from the string. The first parameter is the offset at which to start removing characters. The second parameter is the number of characters to remove. Defaults to removing to the end of the string.
[String->RemoveLeading]	Removes all instances of the parameter from the beginning of the string. Modifies the string and returns no value. Requires a single string parameter.
[String->RemoveTrailing]	Removes all instances of the parameter from the end of the string. Modifies the string and returns no value. Requires a single string parameter.
[String->Replace]	Replaces every occurrence of a substring. Requires two parameters, the substring to find and the replacement string. Modifies the string and returns no value. Optional third parameter specifies the maximum number of replacements to perform.
[String->Reverse]	Reverses the string. Optional parameters specify a character offset and length for a substring to be reversed. Defaults to reversing the entire string. Modifies the string and returns no value.
[String->Trim]	Removes all white space from the start and end of the string. Modifies the string in place and returns no value.

Note: Full documentation and examples for each of the string member tags can be found in the LDML Reference.

To replace a substring:

Use the [String->Replace] tag. The following example replaces every instance of and within the string to or.

```
[Variable: 'myString' = 'Red and Yellow and Blue']
[$myString->(Replace: 'and','or')]
[Output: $myString]
```

→ Red or Yellow or Blue

To remove white space from the start and end of a string:

Use the [String->Trim] tag. The following example removes all the white space from the start and end of the string leaving just the relevant content.

```
[Variable: 'myString' = '    Green and Purple    ']
[$myString->(Trim)]
[Output: $myString]
```

→ Green and Purple

Table 4: String Manipulation Tags

Tag	Description
[String_Concatenate]	Concatenates all of its parameters into a single string.
[String_Insert]	Takes three parameters: a string, a -Text keyword/value parameter which defines the text to be inserted, and a -Position parameter which defines the offset into the string at which to insert the text. Returns a new string with the specified text inserted at the specified location.
[String_Remove]	Takes three parameters: a string, a -StartPosition keyword/value parameter, and a -EndPosition keyword/value parameter. Returns the string with the substring from -StartPosition to -EndPosition removed.
[String_RemoveLeading]	Takes two parameters: a string and a -Pattern keyword/value parameter. Returns the string with any occurrences of the pattern removed from the start.
[String_RemoveTrailing]	Takes two parameters: a string and a -Pattern keyword/value parameter. Returns the string with any occurrences of the pattern removed from the start.
[String_Replace]	Takes three parameters: a string, a -Find keyword/value parameter, and a -Replace keyword/value parameter. Returns the string with the first instance of the -Find parameter replaced by the -Replace parameter.

Note: Full documentation and examples for each of the string tags can be found in the LDML Reference.

Examples of using string manipulation tags:

- The [String_Extract] tag can be used to return a portion of a string. In the following example five characters of the string A Short String are returned
[String_Extract: 'A Short String', -StartPosition=3, -EndPosition=8] → Short

- The [String_Remove] tag is similar, but rather than returning a portion of a string, it removes a portion of the string and returns the remainder. In the following example five characters of the string A Short String are removed and the remainder is returned.

[String_Remove: 'A Short String', -StartPosition=3, -EndPosition=8] → A String

- The [String_RemoveLeading] and [String_RemoveTrailing] tags can be used to remove a repeating character from the start or end of a string. In the following example asterisks are removed from a string *A Short String*.

[String_RemoveLeading: -Pattern='*',
(String_RemoveTrailing: -Pattern='*', *A Short String*)]

→ A Short String

- The [String_Replace] tag can be used to replace a portion of a string with new characters. In the following example the word Short is replaced by the word Long.

[String_Replace: 'A Short String', -Find='Short', -Replace='Long'] → A Long String

Note: For more powerful string manipulation see the *Regular Expressions* section below.

String Conversion Tags

The string data type includes many tags that can be used to change the case of string values. The available member tags are listed in *Table 5: String Conversion Member Tags* and the available substitution tags are listed in *Table 6: String Conversion Tags*.

The member tags in this section all modify the base string in place and do not return a value. In order to see the converted string, the variable containing the string must be output using the [Output] tag.

```
[Variable: 'myString' = 'Test']  
[$myString->(UpperCase)]  
[Output: $myString] → TEST
```

In contrast, the substitution tags return the modified string directly.

```
[String_UpperCase: 'Test'] → TEST
```

The member tags should be used when multiple modifications need to be made to a string that is stored in a variable. The substitution tags can be used when the value is required immediately for output.

Table 5: String Conversion Member Tags

Tag	Description
[String->Foldcase]	Converts all characters in the string for a case-insensitive comparison. Modifies the string and returns no value.
[String->Lowercase]	Converts all characters in the string to lowercase. Modifies the string in place and returns no value. Accepts an optional locale/country code for Unicode conversion.
[String->Titlecase]	Converts the string to titlecase with the first character of each word capitalized. Modifies the string in place and returns no value. Accepts an optional locale/country code for Unicode conversion.
[String->toLower]	Converts a character of the string to lowercase. Requires the position of the character to be modified. Modifies the string in place and returns no value.
[String->toUpper]	Converts a character of the string to uppercase. Requires the position of the character to be modified. Modifies the string in place and returns no value.
[String->toTitle]	Converts a character of the string to titlecase. Requires the position of the character to be modified. Modifies the string in place and returns no value.
[String->Unescape]	Converts a string from the hexadecimal URL encoding.
[String->Uppercase]	Converts all characters in the string to uppercase. Modifies the string in place and returns no value. Accepts an optional locale/country code for Unicode conversion.

Note: Full documentation and examples for each of the string member tags can be found in the LDML Reference.

Table 6: String Conversion Tags

Tag	Description
[String_LowerCase]	Returns the concatenation of all of its parameters in lowercase.
[String_UpperCase]	Returns the concatenation of all of its parameters in lowercase.

Examples of using string conversion tags:

The [String_Uppercase] and [String_Lowercase] tags can be used to alter the case of a string. The following example shows the result after using these tags on the string A Short String.

```
[String_Uppercase: 'A Short String'] → A SHORT STRING
[String_Lowercase: 'A Short String'] → a short string
```

String Validation Tags

The string data type includes many tags that can be used to compare and validate string values. The available member tags are listed in *Table 7: String Validation Member Tags* and the available substitution tags are listed in *Table 8: String Validation Tags*.

All of these tags return a boolean value True or False depending on whether the test succeeds or not.

Table 7: String Validation Member Tags

Tag	Description
[String->BeginsWith]	Returns True if the string begins with the parameter. Comparison is case insensitive. Requires a single string parameter.
[String->Compare]	<p>This tag has three forms. In the first, it returns 0 if the parameter is equal to the string, 1 if the string contains the parameter, and -1 if the string does not contain the parameter. Comparison is case insensitive by default. An optional -Case parameter makes the comparison case sensitive. Requires a single string parameter.</p> <p>The second form requires three parameters. The first two parameters are an offset and length into the third string parameter. The comparison is only performed with this parameter substring.</p> <p>The third form requires two additional parameters. The fourth and fifth parameters are an offset and length into the base string. The comparison is only performed between the base and parameter substrings.</p> <p>[String->CompareCodePointOrder] accepts the same parameters as [String->Compare], but provides accurate comparisons for Unicode characters with code points U+10000 and above.</p>
[String->Contains]	Returns True if the string contains the parameter as a substring. Comparison is case insensitive. Requires a single string parameter.

[String->EndsWith]	Returns True if the string ends with the parameter. Comparison is case insensitive. Requires a single string parameter.
[String->Equals]	Returns True if the parameter of the tag is equal to the string. Comparison is case insensitive. Equivalent to the == symbol.

Note: Full documentation and examples for each of the string member tags can be found in the LDML Reference.

To compare two strings:

Use the string comparison member tags. The following code checks whether a string stored in a variable is equal to or contains another string.

```
[Variable: 'testString' = 'A short string']

[Output: $testString->(BeginsWith: 'a')] → True
[Output: $testString->(BeginsWith: 'A short')] → True
[Output: $testString->(BeginsWith: 'string')] → False
[Output: $testString->(EndsWith: 'string')] → True
[Output: $testString->(Contains: 'short')] → True
[Output: $testString->(Equals: 'a short string')] → True
[Output: $testString->(Compare: 'a short string', -Case)] → False
[Output: $testString->(Compare: 3, 5, 'short')] → True
[Output: $testString->(Compare: 3, 5, 'x short other', 3, 5)] → True
```

Table 8: String Validation Tags

Tag	Description
[String_EndsWith]	Returns boolean True if the string ends with the string specified in the -Find parameter. Takes two parameters: a string value and a -Find keyword/value parameter.

Note: Full documentation and examples for each of the string tags can be found in the LDML Reference.

String Information Tags

The string data type includes many tags that can be used to get information about string and character values. The available member tags are listed in *Table 9: String Information Member Tags* and the available substitution tags are listed in *Table 10: String Information Tags*. In addition, tags

which are specific to getting information about characters in a string are listed in *Table 11: Character Information Member Tags*.

These tags return different data types depending on what information is being retrieved about the string. Those tags that return a character position or require a character position as a parameter all number characters starting from 1 for the first character in the string.

Table 9: String Information Member Tags

Tag	Description
[String->Find]	Returns the position at which the first parameter is found within the string or 0 if the first parameter is not found within the string. Requires a single string parameter.
[String->Get]	Returns a specific character from the string. Requires a single integer parameter.
[String->Size]	Returns the number of characters in the string.[String->Length] is a synonym.
[String->SubString]	Returns a substring. The start of the substring is defined by the first parameter and the length of the substring is defined by the second parameter. Requires two integer parameters.

Note: Full documentation and examples for each of the string member tags can be found in the LDML Reference.

To return the length of a string:

- The length of a string can be returned using the [String->Size] tag.
[Output: 'Alpha'->Size] → 5
- The length of a [Variable] value, [Field] value or any value returned by an LDML tag can be returned using the [String->Size] tag.
[Output: \$StringVariable + ' ' + \$StringVariable->Size] → Alpha 5
[Output: (Field: 'First_Name') + ' ' + (Field: 'First_Name')->Size] → Joe 3

To return a portion of a string:

- A specific character from a string can be returned using the [String->Get] tag. In the following example, the third character of Alpha is returned.
[Output: 'Alpha'->(Get: 3)] → p
- A specific range of characters from a string can be returned using the [String->SubString] tag. In the following example, six characters are returned from the string, starting at the third character.
[Output: 'A String Value'->(Substring: 3, 6)] → String

- The start of a string can be returned using the [String->Substring] tag with the first parameter set to 1. The second parameter will define how many characters are returned from the start of the string. In the following example, the first eight characters of the string are returned.

[Output: 'A String Value'-(Substring: 1, 8)] → A String

- The end of a string can be returned using the [String->Substring] tag with the second parameter omitted. The following example returns the portion of the string after the tenth character.

[Output: 'A String Value'-(Substring: 10)] → Value

Table 10: String Information Tags

Tag	Description
[String_Extract]	Takes three parameters: a string, a -StartPosition keyword/value parameter, and a -EndPosition keyword/value parameter. Returns a substring from -StartPosition to -EndPosition.
[String_FindPosition]	Takes two parameters: a string value and a -Find keyword/value parameter. Returns the location of the -Find parameter in the string parameter.
[String_IsAlpha]	Returns boolean True if the string contains only alphabetic characters (a-z or A-Z).
[String_IsAlphaNumeric]	Returns boolean True if the string contains only alphabetic characters or numerals (a-z, A-Z, or 0-9).
[String_IsDigit]	Returns boolean True if the string contains only numerals (0-9).
[String_IsHexDigit]	Returns boolean True if the string contains only hexadecimal numerals (0-9 and a-f).
[String_IsLower]	Returns boolean True if the string contains only lowercase alphabetic characters (a-z).
[String_IsNumeric]	Returns boolean True if the string contains only numerals, hyphens, or periods.
[String_IsPunctuation]	Returns boolean True if the string contains only punctuation characters.
[String_IsSpace]	Returns boolean True if the string contains only white space.
[String_IsUpper]	Returns boolean True if the string contains only uppercase alphabetic characters (A-Z).
[String_Length]	Returns the number of characters in the string.

Note: Full documentation and examples for each of the string tags can be found in the LDML Reference.

Example of using [String_Length] tag:

The [String_Length] tag can be used to return the number of characters in a string. This tag returns the same results as the [String->Size] tag except the method of calling the tag is somewhat different.

The following example shows how to return the length of the string A Short String using both the [String_Length] tag and the [String->Size] tag. The result in both cases is 14.

```
[String_Length: 'A Short String'] → 14
[Output: 'A Short String'->Size] → 14
```

Examples of using string validation tags:

The characters in a string can be checked to see if they meet certain criteria using the [String_Is...] tags. Each character in the string is checked to see if it meets the criteria of the tag. If any single character does not meet the criteria then False is returned.

- In the following example a string word is checked to see which validation strings it passes. The string is in lowercase and consists entirely of alphabetic characters. It is not in uppercase and does not consist entirely of numeric characters.

```
[String_IsAlpha: 'word'] → True
[String_IsAlphaNumeric: 'word'] → True
[String_IsLower: 'word'] → True
[String_IsNumeric: 'word'] → False
[String_IsUpper: 'word'] → False
```

- In the following example a string 2468 is checked to see which validation strings it passes. The string consists entirely of numeric characters. It does not consist entirely of alphabetic characters.

```
[String_IsAlpha: '2468'] → False
[String_IsAlphaNumeric: '2468'] → True
[String_IsNumeric: '2468'] → True
```

- Some of the validation tags are intended to be used on individual characters. The following example shows how each of these tags can be used.

```
[String_IsDigit: '9'] → True
[String_IsHexDigit: 'a'] → True
[String_IsPunctuation: '!'] → True
[String_IsSpace: ' '] → True
```

Table 11: Character Information Member Tags

Tag	Description
[String->CharDigitValue]	Returns the integer value of a character or -1 if the character is alphabetic. Requires a single parameter that specifies the location of the character to be inspected.
[String->CharName]	Returns the Unicode name of a character. Requires a single parameter that specifies the location of the character to be inspected.
[String->CharType]	Returns the Unicode type of a character. Requires a single parameter that specifies the location of the character to be inspected.
[String->Digit]	Returns the integer value of a character according to a particular radix. Requires two parameters. The first specifies the location of the character to be inspected. The second specifies the radix of the result (e.g. 16 for hexadecimal).
[String->GetNumericValue]	Returns the decimal value of a character or a negative number of the character is alphabetic. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsAlnum]	Returns True if the character is alphanumeric. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsAlpha]	Returns True if the character is alphabetic. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsBase]	Returns True if the character is part of the base characters of Unicode. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsCntrl]	Returns True if the character is a control character. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsDigit]	Returns True if the character is numeric. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsLower]	Returns True if the character is lowercase. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsPrint]	Returns True if the character is printable (i.e. not a control character). Requires a single parameter that specifies the location of the character to be inspected.
[String->IsSpace]	Returns True if the character is a space. Requires a single parameter that specifies the location of the character to be inspected.

[String->IsTitle]	Returns True if the character is titlecase. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsUpper]	Returns True if the character is uppercase. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsWhitespace]	Returns True if the character is white space. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsUAlphabetic]	Returns True if the character has the Unicode alphabetic attribute. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsULowercase]	Returns True if the character has the Unicode lowercase attribute. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsUUppercase]	Returns True if the character has the Unicode uppercase attribute. Requires a single parameter that specifies the location of the character to be inspected.
[String->IsUWhiteSpace]	Returns True if the character has the Unicode white space attribute. Requires a single parameter that specifies the location of the character to be inspected.

Note: Full documentation and examples for each of the string member tags can be found in the LDML Reference.

To inspect the Unicode properties of a string:

Use the character information member tags. The following example shows the information that is provided for a standard ASCII character b. The character name and type are provided according to the Unicode standard. The [String->Integer] tag returns the decimal ASCII value for the character. The [String->Digit] tag with a radix of 16 returns the hexadecimal value for the character.

```
[Output: 'b'->(CharName: 1)] → LATIN SMALL LETTER B
[Output: 'b'->(CharType: 1)] → LOWERCASE_LETTER
[Output: 'b'->(IsLower: 1)] → True
[Output: 'b'->(IsUpper: 1)] → False
[Output: 'b'->(IsWhiteSpace: 1)] → False
[Output: 'b'->(Integer: 1)] → 98
[Output: 'b'->(Digit: 1, 16)] → 11
```

The information tags can be used on any Unicode characters. The following example shows the tags being used on a Traditional Chinese character 並 that roughly translates to “and”. The character is neither uppercase nor lowercase and is identified by the Unicode reference 4E26.

```
[Output: '並'-(CharName: 1)] → CJK UNIFIED IDEOGRAPH-4E26
[Output: '並'-(CharType: 1)] → OTHER_LETTER
[Output: '並'-(IsLower: 1)] → False
[Output: '並'-(IsUpper: 1)] → False
[Output: '並'-(IsWhiteSpace: 1)] → False
```

Note: The character 並 can be represented in a string by `\u4E26` or in HTML as the entity `並`.

Table 12: Unicode Tags

Tag	Description
[String_GetUnicodeVersion]	Returns the version of the Unicode standard which Lasso supports.
[String_CharFromName]	Returns the character corresponding to the specified Unicode character name.

String Casting Tags

The string data type includes many tags which can be used to cast a value to or from the string data type. These tags are documented in the *Casting Values to Strings* section earlier in this chapter and in corresponding sections in the chapters for each data type.

In addition, the [String->Split] tag can be used to cast a string into an array. This tag is described in *Table 13: String Casting Member Tags*.

Table 13: String Casting Member Tags

Tag	Description
[String->Split]	Splits the string into an array of strings based on the delimiter specified in the first parameter. This tag does not modify the string, but returns the new array. Requires a single string parameter.

To convert a string into an array:

A string can be converted into an array using the [String->Split] tag. A single parameter defines what character should be used to split the string into the multiple elements of the array. The following example splits a string on the space character, returning an array of words from the string.

```
[Output: 'A String Value'-(Split: ' ')]
```

→ (Array: (A), (String), (Value))

Regular Expressions

The [String_FindRegExp] and [String_ReplaceRegExp] tags can be used to perform regular expressions find and replace routines on text strings. A regular expression is a powerful pattern-matching language that allows complex replacements to be specified easily.

Note: Full documentation of regular expression methodology is outside the scope of this manual. The implementation of regular expressions in LDML closely matches that in the Perl language. Consult a standard reference on regular expressions for more information about how to use this flexible technology.

Table 14: Regular Expression Tags

Tag	Description
[String_FindRegExp]	Takes two parameters: a string value and a -Find keyword/value parameter. Returns an array with each instance of the -Find regular expression in the string parameter. Optional -IgnoreCase parameter uses case insensitive patterns.
[String_ReplaceRegExp]	Takes three parameters: a string value, a -Find keyword/value parameter, and a -Replace keyword/value parameter. Returns an array with each instance of the -Find regular expression replaced by the value of the -Replace regular expression the string parameter. Optional -IgnoreCase parameter uses case insensitive parameters. Optional -ReplaceOnlyOne parameter replaces only the first pattern match.

A regular expression is assembled by creating a match string. The simplest match string is just a word containing characters or numbers. The match string `bird` matches the word “bird”. Match strings are case sensitive unless the `-IgnoreCase` parameter is specified. Match strings can also contain symbols such as `\w` which matches any alphanumeric character. The match string `\b\wrd` would match the word “bird” or the word “bard”. Square brackets can be used to generate custom sets of characters or ranges of characters. The match string `[bc]ard` will match either the word “bard” or the word “card”. The match string `[bB]ard` will match either the word “bard” or the word “Bard”.

All of the symbols which can be used in match strings are detailed in *Table 15: Regular Expression Matching Symbols*.

Table 15: Regular Expression Matching Symbols

Symbol	Description
a-z A-Z 0-9	Alphanumeric characters (and any other characters not defined as symbols) match the specified character. Case sensitive.
.	Period matches any single character.
^	Circumflex matches the beginning of a line.
\$	Dollar sign matches the end of a line.
\\...	Escapes the next character. This allows any symbol to be specified as a matching character.
[...]	Character class. Matches any character contained within the square brackets.
[^...]	Character exception class. Matches any character which is not contained within the square brackets.
[a-z]	Character range. Matches any character between the two character specified. Can be used with characters or numbers.
\\t	Matches a tab character.
\\r	Matches a return character.
\\n	Matches a new-line character.
\\"	Matches a double quote.
\\'	Matches a single quote.
\\w	Matches an alphanumeric 'word' character (underscore included).
\\W	Matches a non-alphanumeric character.
\\s	Matches a blank, whitespace character (space, tab, carriage return, etc.).
\\S	Matches a non-blank, non-whitespace character.
\\d	Matches a digit character (0-9).
\\D	Matches a non-digit character.

Note: Other than the built-in escaped characters `\\n`, `\\r`, `\\t`, `\\"`, and `\\'` all other escaped characters in regular expressions should be preceded by two backslashes.

Matching symbols can be used as components of more complex expressions using combination symbols. The simplest combination symbol is `+` which matches the preceding matching symbol one or more times. The expression `[abcd]+` matches any word containing only the letters `a`, `b`, or `c` including `"cab"`, `"cad"`, `"dab"`, `"bad"`, `"add"`, etc.

All of the symbols which can be used in match strings are detailed in *Table 16: Regular Expression Combination Symbols*.

Table 16: Regular Expression Combination Symbols

Symbol	Description
	Alternation. Matches either the character before or the character after the symbol.
()	Grouping for output. Defines a named group for output. Nine groups can be defined.
(?:)	Grouping without output. Can be used to create a logical grouping that should not be assigned to an output.
*	Asterisk. Matches 0 or more repetitions of the preceding symbol.
*?	Non-greedy variant works the same as asterisk, but matches the shortest string possible.
+	Plus Sign. Matches 1 or more repetitions of the preceding symbol.
+?	Non-greedy variant works the same as the plus sign, but matches the shortest string possible.
?	Question Mark. Makes the preceding symbol optional.
{n}	Matches n repetitions of the preceding symbol.
{n,}	Matches at least n repetitions of the preceding symbol.
{n,m}	Matches at least n, but no more than m repetitions of the preceding symbol.
{...}?	Non-greedy variant works the same as the bracketed expressions, but matches the shortest string possible.

The parentheses are a special combination symbol that defines a portion of the match string as a named sub-expression that can be referenced in the replacement string. For example a matching string of blue(world) would match the word blueworld. A replacement string of green\1 would then result in greenworld as output. The word world is named as sub-expression 1 by virtue of the parentheses.

Table 17: Regular Expression Replacement Symbols

Symbol	Description
a-z A-Z 0-9	Alphanumeric characters (and any other characters not defined as symbols) place the specified character in the output.
\1 ... \9	Names a group in the replace string. Up to nine groups can be specified using the numerals 1 through 9.

Note: Other than the built-in escaped characters \n, \r, \t, \", and \' all other escaped characters in regular expressions should be preceded by two backslashes.

Table 18: Regular Expression Advanced Symbols

Symbol	Description
(#)	Regular expression comment. The contents are not interpreted as part of the regular expression.
(?i)	Sets the remainder of the regular expression to be case insensitive. Similar to specifying -IgnoreCase.
(?-i)	Sets the remainder of the regular expression to be case sensitive (the default).
(?:)	The contents of this group will be matched case insensitive and the group will not be added to the output.
(?-i:)	The contents of this group will be matched case sensitive and the group will not be added to the output.
(?=)	Positive look ahead assertion. The contents are matched following the current position, but not added to the output pattern.
(?!)	Negative look ahead assertion. The same as above, but the content must not match following the current position.
(?<=)	Positive look behind assertion. The contents are matched preceding the current position, but not added to the output pattern.
(?<!)	Negative look behind assertion. The same as above, but the contents must not match preceding the current position.
\b	Matches the boundary between a word and a space.
\B	Matches a boundary not between a word and a space.

Examples of using [String_ReplaceRegExp]:

The [String_ReplaceRegExp] tag works much like [String_Replace] except that both the -Find parameter and the -Replace can be regular expressions.

- In the following example, every occurrence of the word Blue in the string is replaced by the HTML code Blue so that the word Blue appears in blue on the Web page. The -Find parameter is specified so

either a lowercase or uppercase b will be matched. The -Replace parameter references \1 to insert the actual value matched into the output.

```
[String_ReplaceRegExp: 'Blue Lake sure is blue today.',
  -Find='([Bb]lue)',
  -Replace='<font color="blue">\1</font>', -EncodeNone]
```

→ Blue lake sure is blue today.

- In the following example, every email address is replaced by an HTML anchor tag that links to the same email address. The \w symbol is used to match any alphanumeric characters or underscores. The at sign @ matches itself. The period must be escaped \. in order to match an actual period and not just any character. This pattern matches any email address of the type name@example.com.

```
[String_ReplaceRegExp: 'Send email to documentation@blueworld.com.',
  -Find='(\w+@\w+\.\w+)',
  -Replace='<a href="mailto:\1">\1</a>', -EncodeNone]
```

→ Send email to
 documentation@blueworld.com.

Examples of using [String_FindRegExp]:

The [String_FindRegExp] tag returns an array of items which match the specified regular expression within the string. The array contains the full matched string in the first element, followed by each of the matched subexpressions in subsequent elements.

- In the following example, every email address in a string is returned in an array.

```
[String_FindRegExp: 'Send email to documentation@blueworld.com.',
  -Find='\w+@\w+\.\w+']
```

→ (Array: (documentation@blueworld.com))

- In the following example, every email address in a string is returned in an array and sub-expressions are used to divide the username and domain name portions of the email address. The result is an array with the entire match string, then each of the sub-expressions.

```
[String_FindRegExp: 'Send email to documentation@blueworld.com.',
  -Find='(\w+)(\w+\.\w+)']
```

→ (Array: (documentation@blueworld.com), (documentation), (blueworld.com))

- In the following example, every word in the source is returned in an array. The first character of each word is separated as a sub-expression. The returned array contains 16 elements, one for each word in the source

string and one for the first character sub-expression of each word in the source string.

```
[String_FindRegExp: 'The quick brown fox jumped over a lazy dog.',  
-Find='(\\w)\\w*']
```

→ (Array: (The), (T), (quick), (q), (brown), (b), (fox), (f), (jumped), (j), (over), (o), (a), (a), (lazy), (l), (dog), (d))

The resulting array can be divided into two arrays using the following code. This code loops through the array (stored in `Result_Array`) and places the odd elements in the array `Word_Array` and the even elements in the array `Char_Array` using the `[Repetition]` tag.

```
[Variable: 'Word_Array' = (Array), 'Char_Array'=(Array)]  
[Variable: 'Result_Array' = (String_FindRegExp:  
'The quick brown fox jumped over a lazy dog.', -Find='(\\w)\\w*')]  
[Loop: $Result_Array->Size]  
[If: (Repetition) == 2]  
  [$Char_Array->(Insert: $Result_Array->(Get: (Loop_Count)))]  
[Else]  
  [$Word_Array->(Insert: $Result_Array->(Get: (Loop_Count)))]  
[/If]  
[Loop]  
<br>[Output:$Word_Array]  
<br>[Output: $Char_Array]
```

→
(Array: (The), (quick), (brown), (fox), (jumped), (over), (a), (lazy), (dog))

(Array: (T), (q), (b), (f), (j), (o), (a), (l), (d))

- In the following example, every phone number in a string is returned in an array. The `\\d` symbol is used to match individual digits and the `{3}` symbol is used to specify that three repetitions must be present. The parentheses are escaped `\\(` and `\\)` so they aren't treated as grouping characters.

```
[String_FindRegExp: 'Phone (800) 555-1212 for information.'  
-Find='\\(\\d{3}\\) \\d{3}-\\d{4}']
```

→ (Array: ((800) 555-1212))

- In the following example, only words contained within HTML bold tags ` ... ` are returned. Positive look ahead and look bind assertions are used to find the contents of the tags without the tags themselves. Note that the pattern inside the assertions uses a non-greedy modifier.

```
[String_FindRegExp: 'This is some <b>sample text</b>!'  
-Find='(?:<= <b>).+?(?=</b>)']
```

→ (Array: (sample text))

15

Chapter 15

Math Operations

Numbers in Lasso are stored and manipulated using the decimal and integer data types. This chapter details the symbols and tags that can be used to manipulate decimal and integer values and to perform mathematical operations.

- **Overview** provides an introduction to the decimal and integer data types and how to cast values to and from other data types.
- **Math Symbols** describes the symbols that can be used to create mathematical expressions.
- **Decimal Member Tags** describes the member tags that can be used with the decimal data type.
- **Integer Member Tags** describes the member tags that can be used with the integer data type.
- **Math Tags** describes the substitution and process tags that can be used with numeric values.

Overview

Mathematical operations and number formatting can be performed in LDML using a variety of different methods on integer and decimal values. There are three types of operations that can be performed:

- **Symbols** can be used to perform mathematical calculations within LDML tags or to perform assignment operations within LassoScripts.
- **Member Tags** can be used to format decimal or integer values or to perform bit manipulations.
- **Substitution Tags** can be used to perform advanced calculations.

Each of these methods is described in detail in the sections that follow. This guide contains a description of every symbol and tag and many examples of their use. The LDML Reference is the primary documentation source for LDML symbols and tags. It contains a full description of each symbol and tag including details about each parameter.

Integer Data Type

The integer data type represents whole number values. Basically, any positive or negative number which does not contain a decimal point is an integer value in Lasso. Examples include -123 or 456. Integer values may also contain hexadecimal values such as 0x1A or 0xff.

Spaces must be specified between the + and - symbols and the parameters, otherwise the second parameter of the symbol might be mistaken for an integer literal.

Table 1: Integer Tag

Tag	Description
[Integer]	Casts a value to type integer.

Examples of explicit integer casting:

- Strings which contain numeric data can be cast to the integer data type using the [Integer] tag. The string must start with a numeric value. In the following examples the number 123 is the result of each explicit casting. Only the first integer found in the string 123 and then 456 is recognized.

[Integer: '123'] → 123
[Integer: '123 and then 456'] → 123

- Decimals which are cast to the integer data type are rounded to the nearest integer.

[Integer: 123.000000] → 123
[Integer: 123.999] → 124

Decimal Data Type

The decimal data type represents real or floating point numbers. Basically, any positive or negative number which contains a decimal point is a decimal value in Lasso. Examples include -123.0 and 456.789. Decimal values can also be written in exponential notation as in 1.23e2 which is equivalent to 1.23 times 10² or 123.0.

Spaces must be specified between the + and - symbols and the parameters, otherwise the second parameter of the symbol might be mistaken for a decimal literal.

Table 2: Decimal Tag

Tag	Description
[Decimal]	Casts a value to type decimal.

The precision of decimal numbers is always displayed as six decimal places even though the actual precision of the number may vary based on the size of the number and its internal representation. The output precision of decimal numbers can be controlled using the [Decimal->Format] tag described later in this chapter.

Examples of implicit decimal casting:

- Integer values are cast to decimal values automatically if they are used as a parameter to a mathematical symbol. If either of the parameters to the symbol is a decimal value then the other parameter is cast to a decimal value automatically. The following example shows how the integer 123 is automatically cast to a decimal value because the other parameter of the + symbol is the decimal value 456.0.

[Output: 456.0 + 123] → 579.000000

The following example shows how a variable with a value of 123 is automatically cast to a decimal value.

[Variable: 'Number'=123]

[Output: 456.0 + (Variable: 'Number')] → 579.000000

Examples of explicit decimal casting:

- Strings which contain numeric data can be cast to the decimal data type using the [Decimal] tag. The string must start with a numeric value. In the following examples the number 123.000000 is the result of each explicit casting. Only the first decimal value found in the string 123 and then 456 is recognized.

[Decimal: '123'] → 123.000000

[Decimal: '123.0'] → 123.000000

[Decimal: '123 and then 456'] → 123.000000

- Integers which are cast to the decimal data type simply have a decimal point appended. The value of the number does not change.

[Decimal: 123] → 123.000000

Mathematical Symbols

The easiest way to manipulate integer and decimal values is to use the mathematical symbols. *Table 3: Mathematical Symbols* details all the symbols that can be used with integer and decimal values.

Table 3: Mathematical Symbols

Symbol	Description
+	Adds two numbers. This symbol should always be separated from its parameters by a space.
-	Subtracts the right parameter from the left parameter. This symbol should always be separated from its parameters by a space.
*	Multiplies two numbers.
/	Divides the left parameter by the right parameter.
%	Modulus. Calculates the left parameter modulo the right number. Both parameters must be integers.

Each of the mathematical symbols takes two parameters. If either of the parameters is a decimal value then the result will be a decimal value. Many of the symbols can also be used to perform string operations. If either of the parameters is a string value then the string operation defined by the symbol will be performed rather than the mathematical operation.

Note: Full documentation and examples for each of the mathematical symbols can be found in the LDML Reference.

Examples of using the mathematical symbols:

- Two numbers can be added using the + symbol. The output will be a decimal value if either of the parameters are a decimal value. Note that the symbol + is separated from its parameters by spaces and negative values used as the second parameter should be surrounded by parentheses.

[Output: 100 + 50] → 150
[Output: 100 + (-12.5)] → 87.500000

- The difference between numbers can be calculated using the - symbol. The output will be a decimal value if either of the parameters are a decimal value.

[Output: 100 - 50] → 50
[Output: 100 - (-12.5)] → 112.500000

- Two numbers can be multiplied using the * symbol. The output will be a decimal value if either of the parameters are a decimal value.

[Output: 100 * 50] → 5000

[Output: 100 * (-12.5)] → -1250.000000

Table 4: Mathematical Assignment Symbols

Symbol	Description
=	Assigns the right parameter to the variable designated by the left parameter.
+=	Adds the right parameter to the value of the left parameter and assigns the result to the variable designated by the left parameter.
-=	Subtracts the right parameter from the value of the left parameter and assigns the result to the variable designated by the left parameter.
*=	Multiplies the value of the left parameter by the value of the right parameter and assigns the result to the variable designated by the left parameter.
/=	Divides the value of the left parameter by the value of the right parameter and assigns the result to the variable designated by the left parameter.
%=	Modulus. Assigns the value of the left parameter modulo the right parameter to the left parameter. Both parameters must be integers.

Each of the symbols takes two parameters. The first parameter must be a variable that holds an integer or decimal value. The second parameter can be any integer or decimal value. The result of the operation is calculated and then stored back in the variable specified as the first operator.

Note: Full documentation and examples for each of the mathematical symbols can be found in the LDML Reference.

Examples of using the mathematical assignment symbols:

- A variable can be assigned a new value using the = symbol. The following example shows how to define an integer variable and then set it to a new value. The new value is output using the [Output] tag.

```
<?LassoScript
  Variable: 'IntegerVariable'= 100;
  $IntegerVariable = 123456;
  Output: $IntegerVariable;
?>
```

→ 123456

- A variable can be used as a collector by adding new values using the += symbol. The following example shows how to define an integer variable and then add several values to it. The final value is output using the [Output] tag.

```
<?LassoScript
  Variable: 'IntegerVariable' = 0;
  $IntegerVariable += 123;
  $IntegerVariable += (-456);
  Output: $IntegerVariable;
?>
```

→ -333

Table 5: Mathematical Comparison Symbols

Symbol	Description
==	Returns True if the parameters are equal.
!=	Returns True if the parameters are not equal.
<	Returns True if the left parameter is less than the right parameter.
<=	Returns True if the left parameter is less than or equal to the right parameter.
>	Returns True if the left parameter is greater than the right parameter.
>=	Returns True if the left parameter is greater than or equal to the right parameter.

Each of the mathematical symbols takes two parameters. If either of the parameters is a decimal value then the result will be a decimal value. Many of the symbols can also be used to perform string operations. If either of the parameters is a string value then the string operation defined by the symbol will be performed rather than the mathematical operation.

Note: Full documentation and examples for each of the mathematical symbols can be found in the LDML Reference.

Examples of using the mathematical comparison symbols:

- Two numbers can be compared for equality using the == symbol and != symbol. The result is a boolean True or False. Integers are automatically cast to decimal values when compared.

```
[Output: 100 == 123] → False
[Output: 100.0 != (-123.0)] → True
[Output: 100 == 100.0] → True
[Output: 100.0 != (-123)] → False
```

- Numbers can be ordered using the `<`, `<=`, `>`, and `>=` symbols. The result is a boolean `True` or `False`.

[Output: `-37 > 0`] → `False`

[Output: `100 < 1000.0`] → `True`

Decimal Member Tags

The decimal data type includes one member tag that can be used to format decimal values.

Table 6: Decimal Member Tag

Tag	Description
[Decimal->SetFormat]	Specifies the format in which the decimal value will be output when cast to string or displayed to a visitor.

Note: Full documentation and examples for this tag can be found in the LDML Reference.

Decimal Format

The [Decimal->SetFormat] tag can be used to change the output format of a variable. When the variable is next cast to data type string or output to the format file it will be formatted according to the preferences set in the last call to [Decimal->SetFormat] for the variable. If the [Decimal->SetFormat] tag is called with no parameters it resets the formatting to the default. The tag takes the following parameters.

Table 7: [Decimal->SetFormat] Parameters

Keyword	Description
-Precision	The number of decimal points of precision that should be output. Defaults to 6.
-DecimalChar	The character which should be used for the decimal point. Defaults to a period.
-GroupChar	The character which should be used for thousands grouping. Defaults to empty.
-Scientific	Set to True to force output in exponential notation. Defaults to False so decimals are only output in exponential notation if required.
-Padding	Specifies the desired length for the output. If the formatted number is less than this length then the number is padded.
-PadChar	Specifies the character that will be inserted if padding is required. Defaults to a space.
-PadRight	Set to True to pad the right side of the output. By default, padding is appended to the left side of the output.

To format a decimal number as US currency:

Create a variable that will hold the dollar amount, DollarVariable. Use [Decimal->SetFormat] to set the -Precision to 2 and the -GroupChar to comma.

```
[Variable: 'DollarVariable' = 0.0]
[$DollarVariable->(SetFormat: -Precision=2, -GroupChar=',')]
<br>${Output: $DollarVariable}

[Variable: 'DollarVariable' = $DollarVariable + 1000]
[$DollarVariable->(SetFormat: -Precision=2, -GroupChar=',')]
<br>${Output: $DollarVariable}

[Variable: 'DollarVariable' = $DollarVariable / 8]
[$DollarVariable->(SetFormat: -Precision=2, -GroupChar=',')]
<br>${Output: $DollarVariable}
```

→
\$0.00

\$1,000.00

\$12.50

Integer Member Tags

The integer data type includes many member tags that can be used to format or perform bit operations on integer values. The available member tags are listed in *Table 8: Integer Member Tags*.

Table 8: Integer Member Tags

Tag	Description
[Integer->SetFormat]	Specifies the format in which the integer value will be output when cast to string or displayed to a visitor.
[Integer->BitAnd]	Performs a bitwise And operation between each bit in the base integer and the integer parameter.
[Integer->BitOr]	Performs a bitwise Or operation between each bit in the base integer and the integer parameter.
[Integer->BitXOr]	Performs a bitwise Exclusive-Or operation between each bit in the base integer and the integer parameter.
[Integer->BitNot]	Flips every bit in the base integer.
[Integer->BitShiftLeft]	Shifts the bits in the base integer left by the number specified in the integer parameter.
[Integer->BitShiftRight]	Shifts the bits in the base integer right by the number specified in the integer parameter.
[Integer->BitClear]	Clears the bit specified in the integer parameter.
[Integer->BitFlip]	Flips the bit specified in the integer parameter.
[Integer->BitSet]	Sets the bit specified in the integer parameter.
[Integer->BitTest]	Returns true if the bit specified in the integer parameter is true.

Note: Full documentation and examples for each of the integer member tags can be found in the LDML Reference.

Integer Format

The [Integer->SetFormat] tag can be used to change the output format of a variable. When the variable is next cast to data type string or output to the format file it will be formatted according to the preferences set in the last call to [Integer->SetFormat] for the variable. If the [Integer->SetFormat] tag is called with no parameters it resets the formatting to the default. The tag takes the following parameters.

Table 9: [Integer->SetFormat] Parameters

Keyword	Description
-Hexadecimal	If set to True, the integer will output in hexadecimal notation.
-Padding	Specifies the desired length for the output. If the formatted number is less than this length then the number is padded.
-PadChar	Specifies the character that will be inserted if padding is required. Defaults to a space.
-PadRight	Set to True to pad the right side of the output. By default, padding is appended to the left side of the output.

To format an integer as a hexadecimal value:

Create a variable that will hold the value, HexVariable. Use [Integer->SetFormat] to set -Hexadecimal to True..

```
[Variable: 'HexVariable' = 255]
[$HexVariable->(SetFormat: -Hexadecimal=True)]
<br>[Output: $HexVariable]

[Variable: 'HexVariable' = $HexVariable / 5]
[$HexVariable->(SetFormat: -Hexadecimal=True)]
<br>[Output: $HexVariable]
```

→
0xff

0x33

Bit Operations

Bit operations can be performed within Lasso's 64-bit integer values. These operations can be used to examine and manipulate binary data. They can also be used for general purpose binary set operations.

Integer literals in LDML can be specified using hexadecimal notation. This can greatly aid in constructing literals for use with the bit operation. For example, 0xff is the integer literal 255. The [Integer->SetFormat] tag with a parameter of -Hexadecimal=True can be used to output hexadecimal values.

The bit operations are divided into three categories.

- The [Integer->BitAnd], [Integer->BitOr], and [Integer->BitXOr] tags are used to combine two integer values using the specified boolean operation. In

the following example the boolean Or of 0x02 and 0x04 is calculated and returned in hexadecimal notation.

```
[Var: 'BitSet'=0x02]
[$BitSet->(SetFormat: -Hexadecimal=True)]
[$BitSet->(BitOr: 0x04)]
[Output: $BitSet]
```

→ 0x06

- The [Integer->BitShiftLeft], [Integer->BitShiftRight], and [Integer->BitNot] tags are used to modify the base integer value in place. In the following example, 0x02 is shifted left by three places and output in hexadecimal notation.

```
[Var: 'BitSet'=0x02]
[$BitSet->(SetFormat: -Hexadecimal=True)]
[$BitSet->(BitShift: 3)]
[Output: $BitSet]
```

→ 0x10

- The [Integer->BitSet], [Integer->BitClear], [Integer->BitFlip], and [Integer->BitTest] tags are used to manipulate or test individual bits from an integer value. In the following example, the second bit an integer is set and then tested.

```
[Var: 'BitSet'=0]
[$BitSet->(BitSet: 2)]
[$BitSet->(BitTest 2)]
```

→ True

Math Tags

LDML contains many substitution tags that can be used to perform mathematical functions. The functionality of many of these tags overlaps the functionality of the mathematical symbols. It is recommended that you use the equivalent symbol when one is available.

Additional tags detailed in the section on *Trigonometry and Advanced Math*.

Table 10: Math Tags

Tag	Description
[Math_Abs]	Absolute value. Requires one parameter.
[Math_Add]	Addition. Returns sum of multiple parameters.
[Math_Ceil]	Ceiling. Returns the next higher integer. Requires one parameter.
[Math_ConvertEuro]	Converts between the Euro and other European Union currencies.
[Math_Div]	Division. Divides each of multiple parameters in order from left to right.
[Math_Floor]	Floor. Returns the next lower integer. Requires one parameter.
[Math_Max]	Maximum of all parameters.
[Math_Min]	Minimum of all parameters.
[Math_Mod]	Modulo. Requires two parameters. Returns the value of the first parameter modulo the second parameter.
[Math_Mult]	Multiplication. Returns the value of multiple parameters multiplied together.
[Math_Random]	Returns a random number.
[Math_RInt]	Rounds to nearest integer. Requires one parameter
[Math_Roman]	Converts a number into roman numerals. Requires one positive integer parameter.
[Math_Round]	Rounds a number with specified precision. Requires two parameters. The first value is rounded to the same precision as the second value.
[Math_Sub]	Subtraction. Subtracts each of multiple parameters in order from left to right.

Note: Full documentation and examples for each of the math tags can be found in the LDML Reference.

If all the parameters to a mathematical substitution tag are integers then the result will be an integer. If any of the parameter to a mathematical substitution tag is a decimal then the result will be a decimal value and will be returned with six decimal points of precision.

In the following example the same calculation is performed with integer and decimal parameters to show how the results vary. The integer example returns 0 since 0.125 rounds down to zero when cast to an integer.

```
[Math_Div: 1, 8] → 0
[Math_Div: 1.0, 8] → 0.125000
```

Examples of using math substitution tags:

The following are all examples of using math substitution tags to calculate the results of various mathematical operations.

```
[Math_Add: 1, 2, 3, 4, 5] → 15
[Math_Add: 1.0, 100.0] → 101.000000
[Math_Sub: 10, 5] → 5
[Math_Div: 10, 9] → 11
[Math_Div: 10, 8.0] → 12.500000
[Math_Max: 100, 200] → 200
```

Rounding Numbers

Lasso provides a number of different methods for rounding numbers:

- Numbers can be rounded to integer using the [Math_RInt] tag to round to the nearest integer, the [Math_Floor] tag to round to the next lowest integer, or the [Math_Ceil] tag to round to the next highest integer.

```
[Math_RInt: 37.6] → 38
[Math_Floor: 37.6] → 37
[Math_Ceil: 37.6] → 38
```

- Numbers can be rounded to arbitrary precision using the [Math_Round] tag with a decimal parameter. The second parameter should be of the form 0.01, 0.0001, 0.000001, etc.

```
[Math_Round: 3.1415926, 0.0001] → 3.1416
[Math_Round: 3.1415926, 0.001] → 3.142
[Math_Round: 3.1415926, 0.01] → 3.14
[Math_Round: 3.1415926, 0.1] → 3.1
```

- Numbers can be rounded to an even multiple of another number using the [Math_Round] tag with an integer or decimal parameter.

```
[Math_Round: 1463, 1000] → 1000
[Math_Round: 1463, 500] → 1500
[Math_Round: 1463, 20] → 1460
[Math_Round: 1463, 3] → 1464
```

[Math_Round: 3.1415926, 0.5] → 3.0
[Math_Round: 3.1415926, 0.25] → 3.25
[Math_Round: 3.1415926, 1.000] → 3.000
[Math_Round: 3.1415926, 5.0] → 5.0

- If a rounded result needs to be shown to the user, but the actual value stored in a variable does not need to be rounded then either the [Integer->SetFormat] or [Decimal->SetFormat] tags can be used to alter how the number is displayed. See the documentation of these tags earlier in the chapter for more information.

Random Numbers

The [Math_Random] tag can be used to return a random number in a given range. The result can optionally be returned in hexadecimal notation (for use in HTML color variables).

Note: When returning integer values [Math_Random] will return a maximum 32-bit value. The range of returned integers is approximately between +/- 2,000,000,000.

Table 11: [Math_Random] Parameters

Keyword	Description
-Min	Minimum value to be returned.
-Max	Maximum value to be returned. For integer results should be one greater than maximum desired value.
-Hex	If specified, returns the result in hexadecimal notation.

To return a random integer value:

In the following example a random number between 1 and 99 is returned. The random number will be different each time the page is loaded.

[Math_Random: -Min=1, -Max=100]

→ 55

To return a random decimal value:

In the following example a random decimal number between 0.0 and 1.0 is returned. The random number will be different each time the page is loaded.

[Math_Random: -Min=0.0, -Max=1.0]

→ 0.55342

To return a random color value:

In the following example a random hexadecimal color code is returned. The random number will be different each time the page is loaded. The range is from 16 to 256 to return two-digit hexadecimal values between 10 and FF.

```
<font color="#[Math_Random: -Min=16, -Max=256, -Hex][Math_Random: -Min=16,
-Max=256, -Hex][Math_Random: -Min=16, -Max=256, -Hex]">Color</font>
```

→ Color

Trigonometry and Advanced Math

Lasso provides a number of tags for performing trigonometric functions, square roots, logarithms, and calculating exponents.

Table 12: Trigonometric and Advanced Math Tags

Tag	Description
[Math_ACos]	Arc Cosine. Requires one parameter. The return value is in radians between 0 and π .
[Math_ASin]	Arc Sine. Requires one parameter. The return value is in radians between $-2/\pi$ and $2/\pi$.
[Math_ATan]	Arc Tangent. Requires one parameter. The return value is in radians between $-2/\pi$ and $2/\pi$.
[Math_ATan2]	Arc Tangent of a Quotient. Requires two parameters. The return value is in radians between $-\pi$ and π .
[Math_Cos]	Cosine. Requires one parameter.
[Math_Exp]	Natural Exponent. Requires one parameter. Returns e raised to the specified power.
[Math_Ln]	Natural Logarithm. Requires one parameter. Also [Math_Log].
[Math_Log10]	Base 10 Logarithm. Requires one parameter.
[Math_Pow]	Exponent. Requires two parameters: a base and an exponent. Returns the base raised to the exponent.
[Math_Sin]	Sine. Requires one parameter.
[Math_Sqrt]	Square Root. Requires one positive parameter.
[Math_Tan]	Tangent. Requires one parameter.

Examples of using advanced math substitution tags:

The following are all examples of using math substitution tags to calculate the results of various mathematical operations.

```
[Math_Pow: 3, 3] → 27
[Math_Sqrt: 100.0] → 10.000000
```

Locale Formatting

Lasso can format currency, percentages, and scientific values according to the rules of any country or locale. The tags in *Table 13: Locale Formatting Tags* are used for this purpose. Each tag accepts an optional locale that specifies what rules should be used for the format. The tags default to United States (US) format, but can be set to any 2-letter country code (e.g. FR, UK, MX, etc.).

Table 13: Locale Formatting Tags

Tag	Description
[Currency]	Formats a number as currency. Requires one parameter, the currency amount to format. The second optional parameter specifies the locale for which the currency should be formatted.
[Percent]	Formats a number as a percentage. Requires one parameter, the currency amount to format. The second optional parameter specifies the locale for which the currency should be formatted.
[Scientific]	Formats a number using scientific notation. Requires one parameter, the currency amount to format. The second optional parameter specifies the locale for which the currency should be formatted.
[Locale_Format]	Formats a number. Requires one parameter, the currency amount to format. The second optional parameter specifies the locale for which the currency should be formatted.

16

Chapter 16

Date and Time Operations

Dates and times in Lasso can be stored and manipulated as special date and duration data types. This chapter describes the tags that can be used to manipulate dates and times.

- **Overview** provides an introduction to using the Lasso date and duration data types.
- **Date Tags** describes the substitution and member tags that can be used to cast, format, and display dates and times.
- **Duration Tags** describes the substitution and member tags that can be used to cast, format, and display durations.
- **Date and Duration Math** describes the tags that are used to perform calculations using both dates and durations.

Overview

This chapter introduces the date and the duration data types in LDML 7. Dates are a data type that represent a calendar date and/or clock time. Durations are a data type that represents a length of time in hours, minutes, and seconds. Date and duration data types can be manipulated in a similar manner as integer data types, and operations can be performed to determine date differences, time differences, and more. Date data types may also be formatted and converted to a number of predefined or custom formats, and specific information may be extrapolated from a date data type (day of week, name of month, etc.).

Since dates and durations can take many forms, values that represent a date or a duration must be explicitly cast as date or duration data types using the [Date] and [Duration] tags. For example, a value of 01/01/2002 12:30:00

will be treated as a string data type until it is cast as a date data type using the [Date] tag:

```
[Date:'01/01/2002 12:30:00']
```

Once a value is cast as a date or duration data type, special tags, accessors, conversion operations, and math operations may then be used.

Internal Date Libraries

When performing date operations, Lasso uses internal date libraries to automatically adjust for leap years and daylight saving time for the local time zone in all applicable regions of the world (not all regions recognize daylight saving time). The current time and time zone are based on that of the Web server.

Daylight Saving Time Note: Lasso extracts daylight saving time information from the operating system, and can only support daylight saving time conversions between the years 1970 and 2038. For information on special exceptions with date calculations during daylight saving time, see all the *Date and Duration Math* section.

Date Tags

For Lasso to recognize a string as a date data type, the string must be explicitly cast as a date data type using the [Date] tag.

```
[Date: '5/22/2002 12:30:00']
```

When casting as a date data type using the [Date] tag, the following date formats are automatically recognized as valid date strings by Lasso: These automatically recognized date formats are U.S. or MySQL dates with a four digit year followed by an optional 24-hour time with seconds. The “/”, “-”, and “:” characters are the only punctuation marks recognized in valid date strings by Lasso when used in the formats shown below.

```
1/25/2002
1/25/2002 12:34
1/25/2002 12:34:56
1/25/2002 12:34:56 GMT
```

```
2002-01-25
2002-01-25 12:34:56
2002-01-25 12:34:56 GMT
```

Lasso also recognizes a number of special purpose date formats which are shown below. These are useful when working with HTTP headers or email message headers.

```
20020125T12:34:56
Tue, Dec 17 2002 12:34:56 -0800
Tue Dec 17 12:34:56 PST 2002
```

The date formats which contain time zone information (e.g. -0800 or PST) will be recognized as GMT dates. The time zone will be used to automatically adjust the date/time to the equivalent GMT date/time.

If using a date format not listed above, custom date formats can be defined as date data types using the [Date] tag with the -Format parameter.

The following variations of the automatically recognized date formats are valid without using the -Format parameter.

- If the [Date] tag is used without a parameter then the current date and time are returned. Milliseconds are rounded to the nearest second.
- If the time is not specified then it is assumed to be 00:00:00, midnight on the specified date.

```
mm/dd/yyyy → mm/dd/yyyy 00:00:00
```

- If the seconds are not specified then the time is assumed to be even on the minute.

```
mm/dd/yyyy hh:mm → mm/dd/yyyy hh:mm:00
```

- An optional GMT designator can be used to specify Greenwich Mean Time rather than local time.

```
mm/dd/yyyy hh:mm:ss GMT
```

- Two digit years are assumed to be in the 21st century if they are less than 40 or in the 20th century if they are greater than or equal to 40. Two digit years range from 1940 to 2039. For best results, always use four digit years.

```
mm/dd/00 → mm/dd/2000
mm/dd/39 → mm/dd/2039
mm/dd/40 → mm/dd/1940
mm/dd/99 → mm/dd/1999
```

- Days and months can be specified with or without leading 0s. The following are all valid Lasso date strings.

1/1/02	01/01/02
1/1/2002	01/01/2002
1/1/2002 16:35	01/01/2002 16:35
1/1/2002 16:35:45	01/01/2002 16:35:45
1/1/2002 12:35:45 GMT	01/01/2002 12:35:45 GMT

To cast a value as a date data type:

If the value is in a recognized string format described previously, simply use the [Date] tag.

```
[Date: '05/22/2002'] → 05/22/2002 00:00:00
[Date: '05/22/2002 12:30:00'] → 05/22/2002 12:30:00
[Date: '2002-22-05'] → 2002-22-05 00:00:00
```

If the value is not in a string format described previously, use the [Date] tag with the -Format parameter. For information on how to use the -Format parameter, see the *Formatting Dates* section later in this chapter.

```
[Date: '5.22.02 12:30', -Format='%m.%d.%y %H%M'] → 5.22.02 12:30
[Date: '20020522123000', -Format='%Y%m%d%H%M'] → 200205221230
```

Date values which are stored in database fields or variables can be cast to the date data type using the date tag. The format of the date stored in the field or variable should be in one of the format described above or the -Format parameter must be used to explicitly specify the format.

```
[Date: (Variable: 'myDate')]
[Date: (Field: 'Modified_Date')]
[Date: (Action_Param: 'Birth_Date')]
```


Date Tags

LDML contains date substitution tags that can be used to cast date strings as date data types, format date data types, and perform date/time conversions.

Table 1: Date Substitution Tags

Tag	Description
[Date]	Used to cast values to date data types when used with a valid date string as a parameter. An optional -Format parameter with a date format string may be used to explicitly cast an unknown date format. When no parameter is used, it returns the current date and time. An optional -DateGMT keyword/value parameter returns GMT date and time. Also accepts parameters for -Second, -Minute, -Hour, -Day, -Month, -Year, and -DateGMT for constructing and outputting dates.
[Date_Format]	Changes the output format of a Lasso date. Requires a Lasso date data type or valid Lasso date string as a parameter (auto-recognizes the same formats as the [Date] tag). The -Format keyword/value parameter defines how the date should be reformatted. See the Formatting Dates section below for more information.
[Date_SetFormat]	Sets a date format for output using the [Date] tag for an entire Lasso format file. The -Format parameter uses a format string. An optional -TimeOptional parameter causes the output to not return 00:00:00 if there is no time value.
[Date_GMTToLocal]	Converts a date/time from Greenwich Mean Time to local time of the machine running Lasso Service.
[Date_LocalToGMT]	Converts a date/time from local time to Greenwich Mean Time.
[Date_GetLocalTimeZone]	Returns the current time zone of the machine running Lasso Service as a standard GMT offset string (e.g. -0700). Optional -Long parameter shows the name of the time zone (e.g. PDT).
[Date_Minimum]	Returns the minimum possible date recognized by a Date data type in Lasso.
[Date_Maximum]	Returns the maximum possible date recognized as a Date data type in Lasso.

Note: Full documentation and examples for each date tag can be found in the LDML Reference.

To display date values:

- The current date/time can be displayed with [Date]. The example below assumes a current date of 5/22/2002 14:02:05.
[Date] → 5/22/2002 14:02:05
- The [Date] tag can be used to assemble a date from individual parameters. The following tag assembles a valid Lasso date string by specifying each part of the date separately. Since the time is not specified it is assumed to be midnight on the specified day.
[Date: -Year=2002, -Month=5, -Day=22] → 5/22/2002 00:00:00

To convert date values to and from GMT:

Any date data type can instantly be converted to and from Greenwich Mean Time using the [Date_GMTToLocal] and [Date_LocalToGMT] tags. These tags will only convert the current time zone of the machine running Lasso Service. The following example uses Pacific Time (PDT) as the current time zone.

```
[Date_GMTToLocal:(Date:'5/22/2002 14:02:05')] → 5/22/2002 09:02:05
[Date_LocalToGMT:(Date:'5/22/2002 14:02:05')] → 5/22/2002 07:02:05
```

To show the current time zone for the server running Lasso Service:

The [Date_GetLocalTimeZone] tag displays the current time zone of the machine running Lasso Service. The following example uses Pacific Time (PDT) as the current time zone.

```
[Date_GetLocalTimeZone] → 0700
[Date_GetLocalTimeZone: -Long] → PDT
```

Formatting Dates

The [Date] tag and the [Date_Format] tag each have a -Format parameter which accepts a string of symbols that define the format of the date which should be parsed in the case of the [Date] tag or formatted in the case of the [Date_Format] tag. The symbols which can be used in the -Format parameter are detailed in the following table.

Table 2: Date Format Symbols

Symbol	Description
%D	U.S. date format (mm/dd/yyyy).
%Q	MySQL date format (yyyy-mm-dd).
%r	12-hour time format (hh:mm:ss [AM/PM]).
%T	24-hour time format (hh:mm:ss).
%Y	4-digit year.
%y	2-digit year.
%m	Month number (01=January, 12=December).
%B	Full English month name (e.g. "January").
%b	Abbreviated English month name (e.g. "Jan").
%d	Day of month (01-31).
%w	Day of week (01=Sunday, 07=Saturday).
%A	Full English weekday name (e.g. "Wednesday").
%a	Abbreviated English weekday name (e.g. "Wed").
%H	24-hour time hour (0-23).
%h	12-hour time hour (1-12).
%M	Minute (0-59).
%S	Second (0-59).
%p	AM/PM for 12-hour time.
%G	GMT time zone indicator.
%z	Time zone offset in relation to GMT (e.g. +0100, -0800).
%Z	Time zone designator (e.g. PST, GMT-1, GMT+12)

Each of the date format symbols that returns a number automatically pads that number with 0 so all values returned by the tag are the same length.

- An optional underscore _ between the percent sign % and the letter designating the symbol specifies that space should be used instead of 0 for the padding character (e.g. %_m returns the month number with space padding).

- An optional hyphen - between the percent sign % and the letter designating the symbol specifies that no padding should be performed (e.g. %m returns the month number with no padding).
- A literal percent sign can be inserted using %%.

Note: If the %z or %Z symbols are used when parsing a date, the resulting Lasso date object will represent the equivalent GMT date/time.

To convert Lasso date data types to various formats:

The following examples show how to convert either Lasso date data types or valid Lasso date strings to alternate formats.

```
[Date_Format: '06/14/2001', -Format='%A, %B %d'] → Thursday, June 14
[Date_Format: '06/14/2001', -Format='%a, %b %d'] → Thu, Jun 14
[Date_Format: '2001-06-14', -Format='%Y%m%d%H%M'] → 200106140000

[Date_Format: (Date:'1/4/2002'), -Format='%m.%d.%y'] → 01.04.02
[Date_Format: (Date:'1/4/2002 02:30:00'), -Format='%B, %Y ' ] → January, 2002
[Date_Format: (Date:'1/4/2002 02:30:00'), -Format='%r'] → 2:30 AM
```

To import and export dates from MySQL:

A common conversion in Lasso is converting MySQL dates to and from U.S. dates. Dates are stored in MySQL in the following format yyyy-mm-dd. The following example shows how to import a date in this format to a U.S. date format using the [Date_Format] tag with an appropriate -Format parameter.

```
[Date_Format: '2001-05-22', -Format='%D'] → 5/22/2001
[Date_Format: '5/22/2001', -Format='%Q'] → 2001-05-22

[Date_Format: (Date:'2001-05-22'), -Format='%D'] → 5/22/2001
[Date_Format: (Date:'5/22/2001'), -Format='%Q'] → 2001-05-22
```

To set a custom Lasso date format for a file:

Use the [Date_SetFormat] tag. This allows all date data types on a page to be output in a custom format without the use of the [Date_Format] tag. The format specified is only valid for Lasso code contained in the same file below the [Date_SetFormat] tag.

```
[Date_SetFormat: -Format='%m%d%y']
```

The example above allows the following Lasso date to be output in a custom format without the [Date_Format] tag.

```
[Date:'01/01/2002'] → 010102
```

Date Format Member Tags

In addition to [Date_Format] and [Date_SetFormat], LDML 7 also offers the [Date->Format] member tags for performing format conversions on date data types.

Table 3: Date Format Member Tags

Symbol	Description
[Date->Format]	Changes the output format of a Lasso date data type. May only be used with Lasso date data types. Requires a date format string as a parameter.
[Date->SetFormat]	Sets a date output format for a particular Lasso date data type object. Requires a date format string as a parameter. An optional -TimeOptional parameter causes the output to not return 00:00:00 if there is no time value.

To convert Lasso date data types to various formats:

The following examples show how to convert Lasso date data types to alternate formats using the [Date->Format] tag.

```
[Var:'MyDate'=(Date:'2002-06-14 00:00:00')]
[$MyDate->Format: '%A, %B %d'] → Tuesday, June 14, 2002

[Var:'MyDate'=(Date:'06/14/2002 09:00:00')]
[$MyDate->Format: '%Y%m%d%H%M'] → 200206140900

[Var:'MyDate'=(Date:'01/31/2002')]
[$MyDate->Format: '%d.%m.%y'] → 31.01.02

[Var:'MyDate'=(Date:'09/01/2002')]
[$MyDate->Format: '%B, %Y '] → September, 2002
```

To set an output format for a specific date data type:

Use the [Date->SetFormat] tag. This causes all instances of a particular date data type object to be output in a specified format.

```
[Var:'MyDate'=(Date:'01/01/2002')]
[$MyDate->(SetFormat: '%m%d%y')]
```

The example above causes all instances of [Var:'MyDate'] in the current format file to be output in a custom format without the [Date_Format] or [Date->Format] tag.

```
[Var:'MyDate'] → 010102
```

Date Accessors

A date accessor function returns a specific integer or string value from a date data type, such as the name of the current month or the seconds of the current time. All date accessor tags in LDML 7 are defined in *Table 4: Date Accessor Tags*.

Table 4: Date Accessor Tags

Tag	Description
[Date->Year]	Returns a four-digit integer representing the year for a specified date. An optional -Days parameter returns the number of days in the current year (e.g. 365).
[Date->Month]	Returns the number of the month (1=January, 12=December) for a specified date (defaults to current date). Optional -Long returns the full English month name (e.g. "January") or -Short returns an abbreviated English month name (e.g. "Jan"). An optional -Days parameter returns the number of days in the current month (e.g. 31).
[Date->Day]	Returns the integer day of the month (e.g. 15).
[Date->DayofYear]	Returns integer day of year (out of 365). Will work with leap years as well (out of 366).
[Date->DayofWeek]	Returns the number of the day of the week (1=Sunday, 7=Saturday) for a specified date. Optional -Short returns an abbreviated English day name (e.g. "Sun") and -Long returns the full English day name (e.g. "Sunday").
[Date->Week]	Returns the integer week number for the year of the specified date (out of 52). The -Sunday parameter returns the integer week of year starting from Sunday (default). A -Monday parameter returns integer week of year starting from Monday.
[Date->Hour]	Returns the hour for a specified date/time. An optional -Short parameter returns integer hour from 1 to 12 instead of 1 to 24.
[Date->Minute]	Returns integer minutes from 0 to 59 for a specified date/time.
[Date->Second]	Returns integer seconds from 0 to 59 for the specified date/time.
[Date->Millisecond]	Returns the current integer milliseconds of the current date/time only.
[Date->Time]	Returns the time of a specified date/time.
[Date->GMT]	Returns whether the specified date is in local or GMT time.

To use date accessors:

- The individual parts of the current date/time can be displayed using the [Date->...] tags.

```
[(Date:'5/22/2002 14:02:05')->Year] → 2002
[(Date:'5/22/2002 14:02:05')->Month] → 5
[(Date:'5/22/2002 14:02:05')->(Month: -Long)] → February
[(Date:'5/22/2002 14:02:05')->Day] → 22
[(Date:'5/22/2002 14:02:05')->(DayOfWeek: -Short)] → Wed
[(Date:'5/22/2002 14:02:05')->Time] → 14:02:05
[(Date:'5/22/2002 14:02:05')->Hour] → 14
[(Date:'5/22/2002 14:02:05')->Minute] → 02
[(Date:'5/22/2002 14:02:05')->Second] → 05
```

- The [Date->Millisecond] tag can only return the current number of millisecond value (as related to the clock time) for the machine running Lasso Service.

```
[Date->Millisecond] → 957
```

Duration Tags

A duration is a special data type that represents a length of time. A duration is not a 24-hour clock time, and may represent any number of hours, minutes, or seconds.

Similar to dates, durations must be cast as duration data types before they can be manipulated. This is done using the [Duration] tag. Durations may be cast in an hours:minutes:seconds format, or just as seconds.

```
[Duration:'1:00:00'] → 1:00:00
[Duration:'3600'] → 1:00:00
```

Once a value has been cast as a duration data type, duration calculations and accessors may then be used. Durations are especially useful for calculating lengths of time under 24 hours, although they can be utilized for any lengths of time. Durations are independent of calendar months and years, and durations that equal a length of time longer than one month are only estimates based on the average length of years and months (i.e. 365.2425 days per year, 30.4375 days per month). Duration tags in LDML 7 are summarized in *Table 5: Duration Tags*.

Table 5: Duration Tags

Tag	Description
[Duration]	Casts values as a duration data type. Accepts a duration string for hours:minutes:seconds, or an integer number of seconds. An optional -Week parameter automatically adds a specified number of weeks to the duration. Optional -Day, -Hour, -Minute, and -Second parameters may also be used for automatically adding day, hour, minute, and time increments to the duration.
[Duration->Year]	Returns the integer number of years in a duration (based on an average of 365.25 days per year).
[Duration->Month]	Returns the integer number of months in a duration (based on an average of 30.4375 days per month).
[Duration->Week]	Returns the integer number of weeks in the duration.
[Duration->Day]	Returns the integer number of days in the duration.
[Duration->Hour]	Returns the integer number of hours in the duration.
[Duration->Minute]	Returns the integer number of minutes in the duration.
[Duration->Second]	Returns the integer number of seconds in the duration.

To cast and display durations:

- Durations can be created using the [Duration] tag with the -Week, -Day, -Hour, -Minute, and -Second parameters. This always returns durations in hours:minutes:seconds format.
 - [Duration: -Week=5, -Day=3, -Hour=12] → 924:00:00
 - [Duration: -Day=4, -Hour=2, -Minute=30] → 98:30:00
 - [Duration: -Hour=12, -Minute=45, -Second=50] → 12:45:50
 - [Duration: -Hour=3, -Minute=30] → 03:30:00
 - [Duration: -Minute=15, -Second=30] → 00:15:30
 - [Duration: -Second=30] → 00:00:30
- The -Week, -Day, -Hour, -Minute, and -Second parameters of the [Duration] tag may also be combined with a base duration for ease of use when setting a duration value. This always returns durations in hours:minutes:seconds format.
 - [Duration:'5:30:30', -Week=5, -Day=3, -Hour=12] → 929:30:30
 - [Duration:'1:00:00', -Day=4, -Hour=2, -Minute=30] → 99:30:00
 - [Duration:'3600', -Hour=12, -Minute=45, -Second=50] → 13:45:50

- Specific increments of time can be returned from a duration using the [Duration->...] tags.

```
[(Duration:'8766:30:45')->Year] → 1
[(Duration:'8766:30:45')->Month] → 12
[(Duration:'8766:30:45')->Week] → 52
[(Duration:'8766:30:45')->Day] → 365
[(Duration:'8766:30:45')->Hour] → 8767
[(Duration:'8766:30:45')->Minute] → 525991
[(Duration:'8766:30:45')->Second] → 31559445
```

Date and Duration Math

Date calculations in Lasso can be performed by using special date math tags, durations tags, and math symbols in LDML 7. Date calculations that can be performed include adding or subtracting year, month, week, day, and time increments to and from dates, and calculating time durations. Durations are a new data type that represent a length of time in seconds and are introduced in the preceding *Duration Tags* section.

Daylight Saving Time Note: Lasso does not account for changes to and from daylight saving time when performing date math and duration calculations. One should take this into consideration when performing a date or duration calculation across dates that encompass a change to or from daylight saving time (resulting date may be off by one hour).

Date Math Tags

LDML 7 provides two date math substitution tags for performing date calculations. These tags are generally used for adding increments of time to a date, and output a Lasso date in the format specified. These tags are summarized in *Table 6: Date Math Tags*.

Table 6: Date Math Tags

Tag	Description
[Date_Add]	Adds a specified amount of time to a Lasso date data type or valid Lasso date string. First parameter is a Lasso date. Keyword/value parameters define what should be added to the first parameter: -Millisecond, -Second, -Minute, -Hour, -Day, -Week, -Month, or -Year.
[Date_Subtract]	Subtracts a specified amount of time from a Lasso date data type or valid Lasso date string. First parameter is a Lasso date. Keyword/value parameters define what should be subtracted from the first parameter: -Millisecond, -Second, -Minute, -Hour, -Day, -Week, -Month, or -Year.
[Date_Difference]	Returns the time difference between two specified dates. A duration is the default return value. Optional parameters may be used to output a specific integer time value instead of a duration: -Millisecond, -Second, -Minute, -Hour, -Day, -Week, -Month, -Year. Lasso rounds to the nearest integer when using these optional parameters.

To add time to a date:

A specified number of hours, minutes, seconds, days, or weeks can be added to a date data type or valid date string using the [Date_Add] tag. The following examples show the result of adding different values to the current date 5/22/2002 14:02:05.

```
[Date_Add: (Date), -Second=15] → 5/22/2002 14:02:20
[Date_Add: (Date), -Minute=15] → 5/22/2002 14:17:05
[Date_Add: (Date), -Hour=15] → 5/23/2002 05:02:05
[Date_Add: (Date), -Day=15] → 6/6/2002 14:02:05
[Date_Add: (Date), -Week=15] → 9/4/2002 14:02:05
[Date_Add: (Date), -Month=6] → 11/22/2002 14:02:05
[Date_Add: (Date), -Year=1] → 5/22/2003 14:02:05
```

To subtract time from a date:

A specified number of hours, minutes, seconds, days, or weeks can be subtracted from a date data type or valid date string using the [Date_Subtract] tag. The following examples show the result of subtracting different values from the date 5/22/2001 14:02:05.

```
[Date_Subtract: (Date: '5/22/2001 14:02:05'), -Second=15] → 5/22/2001 14:01:50
[Date_Subtract: (Date:'5/22/2001 14:02:05'), -Minute=15] → 5/22/2001 13:47:05
[Date_Subtract: (Date:'5/22/2001 14:02:05'), -Hour=15] → 5/21/2001 23:02:05
[Date_Subtract: '5/22/2001 14:02:05', -Day=15] → 5/7/2001 14:02:05
[Date_Subtract: '5/22/2001 14:02:05', -Week=15] → 2/6/2001 14:02:05
```

To determine the time difference between two dates:

Use the [Date_Difference] tag. The following examples show how to calculate the time difference between two date data types or valid date strings.

```
[Date_Difference: (Date: '5/23/2002'), (Date:'5/22/2002')] → 24:00:00
[Date_Difference: (Date:'5/23/2002'), (Date:'5/22/2002'), -Second] → 86400
[Date_Difference: (Date:'5/23/2002'), '5/22/2002', -Minute] → 3600
[Date_Difference: (Date: '5/23/2002'), '5/22/2002', -Hour] → 24
[Date_Difference: '5/23/2002', (Date:'5/22/2002'), -Day] → 1
[Date_Difference: '5/23/2002', (Date:'5/30/2002'), -Week] → 1
[Date_Difference: '5/23/2002', '6/23/2002', -Month] → 1
[Date_Difference: '5/23/2002', '5/23/2001', -Year] → 1
```

Date and Duration Math Tags

LDML 7 provides three member tags that perform date math operations requiring both date and duration data types. These tags are used for adding durations to dates, subtracting a duration from a date, and determining a duration between two dates. These tags are summarized in *Table 7: Date and Duration Math Tags*.

Table 7: Date and Duration Math Tags

Tag	Description
[Date->Add]	Adds a duration to a Lasso date data type. Optional keyword/value parameters may be used in place of a duration to define what should be added to the first parameter: -Millisecond, -Second, -Minute, -Hour, -Day, -Week.
[Date->Subtract]	Subtracts a duration from a Lasso date data type. Optional keyword/value parameters may be used in place of a duration to define what should be subtracted from the first parameter: -Millisecond, -Second, -Minute, -Hour, -Day, -Week.

[Date->Difference]	Calculates the duration between two date data types. The second parameter is subtracted from the first parameter to determine a duration. Optional parameters may be used to output a specified integer time value instead of a duration: -Millisecond, -Second, -Minute, -Hour, -Day, -Week, -Month, -Year. Lasso rounds to the nearest integer when using these optional parameters.
--------------------	--

Note: The [Date->Add] and [Date->Subtract] tags do not directly output values, but can be used to change the values of variables that contain date or duration data types.

To add a duration to a date:

Use the [Date->Add] tag. The following examples show how to add a duration to a date and return a date.

```
[Var_Set:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Add:(Duration:'24:00:00'))]
[$MyDate] → 5/23/2002 00:00:00
```

```
[Var_Set:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Add:(Duration:'3600'))]
[$MyDate] → 5/22/2002 12:30:00
```

```
[Var_Set:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Add: -Week=1)]
[$MyDate] → 5/29/2002 00:00:00
```

To subtract a duration from a date:

Use the [Date->Subtract] tag. The following examples show how to subtract a duration from a date and return a date.

```
[Var_Set:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Subtract:(Duration:'24:00:00'))]
[$MyDate] → 5/21/2002
```

```
[Var:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Subtract:(Duration:'7200'))]
[$MyDate] → 5/22/2002 9:30:00
```

```
[Var:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Subtract: -Day=3)]
[$MyDate] → 5/19/2002 00:00:00
```

To determine the duration between two dates:

Use the [Date->Difference] tag. The following examples show how to calculate the time difference between two dates and return a duration.

```
[Var_Set:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Difference:(Date:'5/15/2002 01:30:00'))] → 169:30:00

[Var:'MyDate'=(Date: '5/22/2002')]
[$MyDate->(Difference:(Date:'5/15/2002'), -Day)] → 7
```

Using Math Symbols

In LDML 7, one has the ability to perform date and duration calculations using math symbols (similar to integer data types). If a date or duration appears to the left of a math symbol then the appropriate math operation will be performed and the result will be a date or duration as appropriate. All math symbols that can be used with dates or durations are shown in *Table 8: Date Math Symbols*.

Table 8: Date Math Symbols

Tag	Description
+	Used for adding a date and a duration, or adding two durations.
-	Used for subtracting a duration from a date, subtracting a duration from a duration, or determining the duration between two dates.
*	Used for multiplying durations by an interger value.
/	Used for dividing durations by an integer or duration value.

To add or subtract dates and durations:

The following examples show addition and subtraction operations using dates and durations.

```
[Output: (Date: '5/22/2002') + (Duration:'24:00:00')] → 5/23/2002
[Output: (Date: '5/22/2002') - (Duration:'48:00:00')] → 5/20/2002
```

To determine the duration between two dates:

The following calculates the duration between two dates using the minus symbol (-) .

```
[Output: (Date: '5/22/2002') - (Date:'5/15/2002')] → 168:00:00
```

To add one day to the current date:

The following example adds one day to the current date.

```
[(Date) + (Duration: -Day=1)]
```

To multiply or divide a durations by an integer:

The following examples show multiplication and division operations using durations and integers.

[Output: (Duration: -Minute=10) * 12] → 02:00:00

[Output: (Duration: '60') * 10] → 00:10:00

[Output: (Duration: -Hour=1) / 2] → 00:30:00

[Output: (Duration: '00:30:00') / 10] → 00:03:00

To divide a duration by a duration:

The following examples show division of durations by durations. The resulting value is a decimal data type.

[Output: (Duration: -Hour=24) / (Duration: -Hour=6)] → 4.0

[Output: (Duration: '05:00:00') / (Duration: '00:30:00')] → 10.0

To return the duration between the current date and a day in the future:

The following example returns the duration between the current date and 12/31/2004.

[Output: (Date: '12/31/2004') - (Date)]

17

Chapter 17

Arrays and Maps

This chapter describes the array, map, and pair data types in LDML that allow sets of data to be stored and manipulated.

- *Overview* provides an introduction to arrays, maps, and pairs.
- *Arrays* describes the array data type and its member tags.
- *Maps* describes the map data type and its member tags.
- *Pair* describes the pair data type and its member tags.
- *Common Arrays and Maps* describes substitution tags in LDML that return array or map values.

Overview

Arrays, maps, and pairs are compound data types that allow many values to be stored in a single variable. Each is suited to storing a different type of structured data.

- **Arrays** are used to store a sequence of values. Values are stored and retrieved based on a numeric index. The order of values within the array is preserved.
- **Maps** are used to store and retrieve values based on a key of any type. The order of values within maps are not preserved. Usually, maps are used to store and retrieve values based on a string key.
- **Pairs** are used to store two values in an ordered pair. Either the first or second value can be retrieved. Pairs are most commonly used as values within an array or when retrieving parameters in custom tags.

Arrays

An array is a sequence of values which are stored and retrieved by numeric index. The values stored in an array can be of any data type in LDML. Arrays can store any values from strings and integers to other arrays and maps. By nesting compound data types very complex data structures can be created.

Types of Arrays

Arrays can be used in LDML for several different purposes. The same member tags can be used on each type of array, but some have specific uses when used with a particular type of array. These specific uses are described in the examples for each member tag.

- A **List Array** is a sequence of string, decimal, or integer values. New values can be appended to the end of the list or inserted between two elements of the list using [Array->Insert]. Two lists can be merged using [Array->Merge]. The order of elements in the array is important, but may be manipulated using the array member tags.
- A **Storage Array** is a sequence of “cubby holes” for values. Values are stored into a slot identified by an integer and later retrieved. The [Array->Get] tag is used to store and retrieve values, but the order of elements in the array is never altered and multiple arrays are never merged.
- A **Pair Array** is a sequence of pairs. [Action_Params] returns an array of pairs which identify the command tags and name/value pairs that comprise the current Lasso action. This array can be manipulated and then passed as a parameter to an [Inline] tag.

Creating Arrays

Arrays are created using the [Array] constructor tag. The parameters of the tag become the initial values stored in the array. The parameters can be string, decimal, or integer literals, constructor tags for other complex data types, or name/value pairs which are interpreted as pairs to be added to the array.

Table 1: Array Tag

Tag	Description
[Array]	Creates an array that contains each of the parameters of the tag. If no parameters are specified, an empty array is created.

To create an array:

- The following example creates an empty array and stores it in a variable.

```
[Variable: 'EmptyArray' = (Array)]
```

- The following example shows an array of string literals.

```
[Array: 'String One', 'String Two', 'String Three']
```

- The following example shows an array with a combination of string, decimal, and integer literals.

```
[Array: 'String One', 2, 3.333333]
```

- The following example shows how to use values from database fields, form parameters, variables, or tokens as the initial values for an array.

```
[Array: (Field: 'Field_Name'), (Action_Param: 'Parameter_Name'),  
(Variable: 'Variable_Name'), (Token_Value: 'Token_Name')]
```

- The following example shows an array of pairs. Each name/value pair becomes a single pair within the array returned by the tag.

```
[Array: 'Name_One'='Value_One', 'Name_Two'='Value_Two']
```

- The following example shows an array of arrays. The array returned by the following code will only contain two array elements. Each array element will in turn contain two integer elements. Nested arrays can be used to store mathematical multi-dimensional arrays.

```
[Array: (Array: 1, -1), (Array: -1, 0)]
```

- The following example shows how to create an array from a string. The [String->Split] tag can be used to split a string into an array which contains one element for each substring delimited by the parameter to the tag. The following string is split on the comma , character into an array of four elements.

```
[Output: 'One,Two,Three,Four,Five'->Split(',')] ]
```

Values are always copied into an array. They are never stored by reference to the original value. This applies both to simple data types and compound data types. There is no way in LDML to store a reference to a compound data type, except for the name of the variable containing the data type.

Array Member Tags

The array data type has a number of member tags that can be used to store, retrieve or delete array elements or to otherwise manipulate array values.

Table 2: Array Member Tags

Tag	Description
[Array->Find]	Returns an array of elements that match the parameter. Accepts a single parameter of any data type.
[Array->FindIndex]	Returns an array of the indices for elements that match the parameter. Accepts a single parameter of any data type.
[Array->Get]	Returns an item from the array. Accepts a single integer parameter identifying the index of the item to be returned. This tag can be used as the left parameter of an assignment operator to set an element of the array.
[Array->Insert]	Inserts a value into the array. Accepts a single parameter which is the value to be inserted and an optional integer parameter identifying the index of the location where the value should be inserted. Defaults to the end of the array. Returns no value.
[Array->Join]	Joins the items of the array into a string. Accepts a single string parameter which is placed inbetween each item from the array. The opposite of [String->Split].
[Array->Last]	Returns the last item in the array.
[Array->Merge]	Merges an array parameter into the array. Accepts an array parameter and three integer parameters that identify which items from the array parameter should be inserted into the array. Defaults to inserting the entire array parameter at the end of the array. Returns no value.
[Array->Remove]	Removes an item from the array. Accepts a single integer parameter identifying the index of the item to be removed. Defaults to the last item in the array. Returns no value.
[Array->RemoveAll]	Removes any elements that match the parameter from the array. Accepts a single parameter of any data type. Returns no value.
[Array->Size]	Returns the number of elements in the array.
[Array->Sort]	Reorders the elements of the array in alphabetical or numerical order. Accepts a single boolean parameter. Sorts in ascending order by default or if the parameter is True and in descending order if the parameter is False.

The following examples show how to manipulate an array by getting, setting, inserting, and deleting values. The examples are all based on the following array which contains the seven days of the week in English.

```
[Variable: 'DaysOfWeek' = (Array: 'Sunday', 'Monday', 'Tuesday', 'Wednesday',
  'Thursday', 'Friday', 'Saturday')]
```

To get the size of an array:

Use the `[Array->Size]` tag. The following example shows how to output the size of the `DaysOfWeek` array.

```
[Output: $DaysOfWeek->Size] → 7
```

To get elements of an array:

- To get an element of the array use the `[Array->Get]` tag with the appropriate index. In the following example different elements of the `DaysOfWeek` array are returned.

```
[Output: $DaysOfWeek->(Get: 1)] → Sunday
```

```
[Output: $DaysOfWeek->(Get: 4)] → Wednesday
```

- The last element of the array can be returned by using `[Array->Get]` with a parameter of `[Array->Size]`. `[Array->Size]` will return 7 since the array `DaysOfWeek` is 7 elements long and element 7 of the array is Saturday.

```
[Output: $DaysOFWeek->(Get: ($DaysOfWeek->Size))] → Saturday
```

- All of the elements in the array can be returned using `[Iterate] ... [/Iterate]` tags. The following example shows how to list all of the days of the week.

```
[Iterate: $DaysOfWeek, (Variable: 'DayName')]
```

```
<br>[Variable: 'DayName']
```

```
[/Iterate]
```

```
→ <br>Sunday
   <br>Monday
   <br>Tuesday
   <br>Wednesday
   <br>Thursday
   <br>Friday
   <br>Saturday
```

- Alternately, all of the elements in the array can be returned using `[Loop] ... [/Loop]` tags. The following example shows how to list all of the days of the week by using `[Array->Get]` with a parameter of `[Loop_Count]`.

```
[Loop: ($DaysOfWeek->Size)]
```

```
<br>[Output: $DaysOFWeek->(Get: (Loop_Count))]
```

```
[/Loop]
```

```
→ <br>Sunday
   <br>Monday
   <br>Tuesday
   <br>Wednesday
   <br>Thursday
   <br>Friday
   <br>Saturday
```

To set elements of an array:

The [Array->Get] member tag can be used on the left side of an assignment operator to set the value stored in the specified index within the array.

- In the following example, the value of the second element of the array DaysOfWeek is set to the Spanish word for Monday, Lunes.

```
<?LassoScript
  $DaysOfWeek->(Get: 2) = 'Lunes';
?>
```

The value of the second element of the array can then be output using the [Array->Get] tag.

```
[Output: $DaysOfWeek->(Get: 2)] → Lunes
```

- Elements of the array can be modified using any of the assignment symbols. In the following example, the substring day is removed from the third element of the array using the deletion assignment symbol -= leaving Tues. This value is then output.

```
<?LassoScript
  $DaysOfWeek->(Get: 3) -= 'day';
  Output: $DaysOfWeek->(Get: 3);
?>
```

```
→ Tues
```

To insert elements into an array:

- The [Array->Insert] tag can be used to insert a single element in the array. In the following example Sunday is inserted at the end of the array DaysOfWeek. The whole array is then output.

```
<?LassoScript
  $DaysOfWeek->(Insert: 'Sunday');

  Loop: ($DaysOfWeek->Size);
    Output: $DaysOfWeek->(Get: (Loop_Count)) + ' ';
  /Loop;
?>
```

```
→ Sunday Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

- The [Array->Insert] tag can also be used to insert a single element anywhere in the array. In the following example Tuesday is inserted as the third element of the array DaysOfWeek. This pushes back all the other elements of the array. No values in the array are removed or replaced by the [Array->Insert] tag. The whole array is then output.

```
<?LassoScript
    $DaysOfWeek->(Insert: 'Tuesday', 3);

    Loop: ($DaysOfWeek->Size);
        Output: $DaysOfWeek->(Get: (Loop_Count)) + ' ';
    /Loop;
?>
```

→ Sunday Monday Tuesday Tuesday Wednesday Thursday Friday Saturday

To remove elements from an array:

- The [Array->Remove] tag can be used to remove a single element from the array. If no parameter is specified then the last item of the array is removed. In the following example the last item of the array Saturday is removed and then the entire array is displayed.

```
<?LassoScript
    $DaysOfWeek->(Remove);

    Loop: ($DaysOfWeek->Size);
        Output: $DaysOfWeek->(Get: (Loop_Count)) + ' ';
    /Loop;
?>
```

→ Sunday Monday Tuesday Wednesday Thursday Friday

- The [Array->Remove] tag can also be used to remove a single element anywhere in the array. In the following example the fourth value in the array is removed. This removes the element Wednesday. The whole array is then output.

```
<?LassoScript
    $DaysOfWeek->(Remove: 4);

    Loop: ($DaysOfWeek->Size);
        Output: $DaysOfWeek->(Get: (Loop_Count)) + ' ';
    /Loop;
?>
```

→ Sunday Monday Tuesday Thursday Friday Saturday

To display the elements of an array:

- Arrays can be displayed by simply outputting the variable that contains the array. All of the elements of the array are displayed surrounded by

parentheses. This is useful primarily for debugging purposes so the values in an array can be inspected without writing a loop to output all of the elements of the array.

```
[Variable: 'DaysOfWeek']
```

```
→ (Array: (Sunday), (Monday), (Tuesday), (Wednesday), (Thursday), (Friday),  
(Saturday))
```

- Arrays can be displayed by joining the elements of the array into a string. In the following example the days of the week are output with commas between each element.

```
[Output $DaysOfWeek->(Join: ', ')]
```

```
→ Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

Arrays and Strings

Arrays can be used for string manipulation using a combination of array and string member tags. First, the string to be manipulated is transformed into an array using the [String->Split] tag, then the array is manipulated, and finally the string is rendered from the array using the [Array->Join] tag.

The following example demonstrates how to modify a URL which is stored in a variable. This same technique can be used to modify any string which can be split into array elements based on a specific delimiter.

To parse, modify, and reassemble a URL using array tags:

- 1 Store the URL to be modified in a string variable, here named `URL_Variable`.

```
[Variable: 'URL_Variable' = 'http://www.example.com/default.lasso?  
-FindAll&-Database=Contacts&-Table=People&-KeyField=ID']
```

- 2 Use [String->Split] to break the URL apart into several different variables. First, the string is split on `?` to split the base of the URL from the parameters. These two parameters are stored in temporary variables, `URL_Base` and `URL_Parameters`.

```
[Variable: 'Temp_Array' = ($URL_Variable->(Split: '?'))]  
[Variable: 'URL_Base' = ($Temp_Array->(Get: 1))]  
[Variable: 'URL_Parameters' = ($Temp_Array->(Get: 2))]
```

- 3 Use [String->Split] to break the URL parameters apart into an array at the ampersand `&` character.

```
[Variable: 'URL_Array' = ($URL_Parameters->(Split: '&'))]
```

- 4 Now the parameters array can be manipulated. For this example we will sort it using the [Array->Sort] command. Other options include removing

or inserting elements, merging two or more URL parameter arrays, checking for the existence of specific values, etc.

```
[(URL_Array->Sort)]
```

- 5 Reassemble the URL parameters using the [Array->Join] tag to append each item in the array to a new variable URL_Parameters. An ampersand & is placed between each element of the array.

```
[Variable: 'URL_Parameters'=$URL_Array->(Join: '&')]
```

- 6 Reassemble the full URL by concatenating the original URL_Base to the new URL_Parameters and store the result in URL_Variable.

```
[Variable: 'URL_Variable' = $URL_Base + '?' + $URL_Parameters]
```

- 7 Display the modified URL to confirm that the modifications have been made correctly. The command tags in the URL are now sorted alphabetically.

```
[Variable: 'URL_Variable']
```

→ [http://www.example.com/default.lasso?
-Database=Contacts&-FindAll&-KeyField=ID&-Table=People](http://www.example.com/default.lasso?-Database=Contacts&-FindAll&-KeyField=ID&-Table=People)

Merging Arrays

The [Array->Merge] tag can be used to merge two arrays by placing the elements of the tag's array parameter into the base array. The [Array->Merge] accepts a number of parameters as detailed in *Table 3: [Array->Merge] Parameters*.

Table 3: [Array->Merge] Parameters

Parameter	Description
First	The array which is to be merged; the source array.
Second	The index in the destination array where the elements of the source array should be inserted. Optional, defaults to the end of the destination array.
Third	The index in the source array of the first element which should be inserted into the destination array. Optional, defaults to 1.
Fourth	The number of elements from the source array to insert into the destination array. Optional, defaults to all elements from the third parameter to the end of the source array.

The four parameters to [Array->Merge] allow for a selected subset of the source array to be placed at any location in the destination array. This allows very complex array manipulations to be performed.

To append an array to the end of another array:

Use the [Array->Merge] tag with a single array parameter. All the elements of the array parameter will be inserted at the end of the base array. In the following example, two arrays are created, each containing three integers. The elements of the second array are merged into the elements of the first array and then all the elements of the new array are displayed.

```
<?LassoScript
  Variable: 'First_Array' = (Array: 1, 2, 3);
  Variable: 'Second_Array' = (Array: 4, 5, 6);

  $First_Array->(Merge: $Second_Array);

  Output: $First_Array;
?>
```

→ (Array: (1), (2), (3), (4), (5), (6))

To insert a single element from one array into another array:

In the following example the third element of the Second_Array is inserted as the new first element of the First_Array using the [Array->Merge] tag. The second parameter to [Array->Merge] is set to 1 so the element will be inserted as the first element of First_Array. The third parameter is set to 3 so the third element of Second_Array will be selected. The fourth parameter is set to 1 so only one element of Second_Array will be copied.

```
<?LassoScript
  Variable: 'First_Array' = (Array: 1, 2, 3);
  Variable: 'Second_Array' = (Array: 4, 5, 6);

  $First_Array->(Merge: $Second_Array, 1, 3, 1);

  Output: $First_Array;
?>
```

→ (Array: (6), (1), (2), (3))

Finding Elements of an Array

The [Array->Find] tag can be used to return a subset of an array which matches a specified value. This can be used to determine whether an array contains a value or, when used in concert with the [Array->RemoveAll] tag this can be used to extract a number of elements from an array.

To determine whether an array contains a value:

In the following example the array `DaysOfWeek` is checked to see if it contains an element `Thursday`. `[Array->Find]` returns an array with the elements found in the source array. `[Array->Find]` will return the empty array `[Array]` if the element was not found in the array.

```
<?LassoScript
  Variable: 'DaysOfWeek' = (Array: 'Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday');

  If: ($DaysOfWeek->(Find: 'Thursday') != (Array));
    Output: 'The array contains Thursday!';
  /If;
?>
```

→ The array contains Thursday!

Note: Alternately, the size of the returned array could be checked using `($DaysOfWeek->(Find: Thursday)->Size > 0)`. This would return the same results as the example above.

To find the indices where an element occurs within the array:

In the following example, an array is returned that reports the index of each occurrence of 1 within the array.

```
<?LassoScript
  Variable: 'Find_Array' = (Array: 6, 1, 4, 1, 5, 1, 2, 3, 1);

  Output: $Find_Array->(FindIndex: 1);
?>
```

→ (Array: (2), (4), (6), (9))

The result array can be used to modify each occurrence of 1 within the array. In the following example each occurrence of 1 within the array is changed to 0.

```
<?LassoScript
  Variable: 'Find_Array' = (Array: 6, 1, 4, 1, 5, 1, 2, 3, 1);

  Variable: 'Temp_Array' = $Find_Array(FindIndex: 1);

  Iterate: $Temp_Array, (Variable: 'Temp_Index');
    $Find_Array->(Get: $Temp_Index) = 0;
  /Iterate;

  Output: $Find_Array;
?>
```

→ (Array: (6), (0), (4), (0), (5), (0), (2), (3), (0))

To delete elements with a certain value from an array:

In the following example, all elements with value 1 are deleted from an array `Delete_Array`. The initial array contains many different integer values. The resulting array is output after all the elements with value 1 have been deleted.

```
<?LassoScript
  Variable: 'Delete_Array' = (Array: 6, 1, 4, 1, 5, 1, 2, 3, 1);

  $Delete_Array->(RemoveAll: 1);

  Output: $Delete_Array;
?>
```

→ (Array: (6), (4), (5), (2), (3))

Pair Arrays

Pair arrays can be used to store a sequence of name/value pairs. The order of elements within a pair array is maintained. The `[Action_Params]` and `[Params]` tags both return pair arrays which contain the parameters passed with the current Lasso action or into a custom tag respectively.

To create a pair array:

Use the `[Array]` tag with name/value parameters. Each name/value parameter becomes a pair in the resulting array. The following example shows an array created with three pair elements.

```
[Array: 'Name_One'='Value_One',
      'Name_Two'='Value_Two',
      'Name_Three'='Value_Three']
```

To find pairs within a pair array:

The `[Array->Find]` tag can be used to find pairs within a pair array. The parameter passed to the `[Array->Find]` tag is only compared to the `[Pair->First]` element of each pair. The `[Array->Find]` tag returns an array that contains only the pairs whose first part matches the parameter. The following example shows an array defined with three pair elements. The `[Array->Find]` tag is used to return both elements for the name Alpha.

```
[Variable: 'Pair_Array' = (Array: 'Alpha'='One', 'Beta'='Two', 'Alpha'=1, 'Beta'=2)]
[Output: $Pair_Array->(Find: 'Alpha')]
```

→ (Array: (Pair: (Alpha)=(One)), (Pair: (Alpha)=(1)))

To insert pairs into a pair array:

Use the [Array->Insert] tag with a name/value parameter. The new element will be inserted at the end of the array by default. The following example inserts a new element Gamma=Three into Pair_Array.

```
<?LassoScript
  Variable: 'Pair_Array' = (Array: 'Alpha'='One', 'Beta'='Two', 'Alpha'=1, 'Beta'=2);
  $Pair_Array->(Insert: 'Gamma'='Three');
?>
```

Sorting Arrays

Arrays can be sorted using the [Array->Sort] tag. This tag reorders the elements of the array so they will no longer be available at the index they were originally set.

Examples of sorting arrays:

- The following LassoScript shows an array with integer elements. The array is sorted and then the values of the array are output. The default sort order is ascending.

```
<?LassoScript
  Variable: 'Sort_Array' = (Array: 6, 4, 5, 2, 3, 1);

  $Sort_Array->(Sort);

  Output: $Sort_Array;
?>
```

→ (Array: (1), (2), (3), (4), (5), (6))

- The following LassoScript shows the DaysOfWeek array being sorted in descending alphabetical order. The [Array->Sort] tag accepts one parameter. True for ascending order or False for descending order. The default is True.

```
<?LassoScript
  Variable: 'DaysOfWeek' = (Array: 'Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday');

  $DaysOfWeek->(Sort: False);

  Output: $DaysOfWeek;
?>
```

→ (Array: (Wednesday), (Tuesday), (Thursday), (Sunday), (Saturday), (Monday), (Friday))

Maps

Maps store and retrieve values based on a key. This allows for specific values to be stored under a name and then retrieved later using that same name. The name or key is usually a string value, but can actually be a value of any valid data type in LDML.

Maps can only store one value per key. When a new value with the same key is inserted into a map it replaces the previous value which was stored in the map. If you need to create a data structure that stores more than one value per key, use an array of pairs instead.

Note: The order of elements in a map is not defined. As more elements are added to a map the order may change and should never be relied upon.

Table 4: Map Tag

Tag	Description
[Map]	Creates a map that contains each of the name/value parameters of the tag. If no parameters are specified, an empty map is created.

To create a map:

- The following example creates an empty map and stores it in a variable.
[Variable: 'EmptyMap' = (Map)]
- The following example shows a map with data stored using string literals as keys. The map is similar to a database record storing information about a particular site visitor.
[Map: 'First_Name'='John', 'Last_Name'='Doe', 'Phone_Number'='800-555-1212']
- The following example shows a map with integer literals as keys. This map could be used to lookup the name of a day of the week based on its order within the week.
[Map: 1='Sunday',
2='Monday',
3='Tuesday',
4='Wednesday',
5='Thursday',
6='Friday',
7='Saturday']
- The following example shows a map which contains arrays that are retrieved using string literals as keys. The map contains two arrays which are named Array_One and Array_Two.

```
[Map: 'Array_One' = (Array: 1, 2, 3, 4, 5),
'Array_Two' = (Array: 9, 8, 7, 6, 5)]
```

Map Member Tags

The map data type has a number of member tags that can be used to store, retrieve or delete map elements by key.

Table 5: Map Member Tags

Tag	Description
[Map->Find]	Returns a value from the map by key. Accepts a single parameter which is the key of the value to be returned.
[Map->Get]	Returns a pair from the map by integer index. Accepts a single parameter which is the index of the value to be returned.
[Map->Keys]	Returns an array of all the keys specified in the map.
[Map->Insert]	Inserts a value into the map by key. Accepts a single name/value pair parameter which specifies the key and value to be inserted.
[Map->Remove]	Removes a value from the map by key. Accepts a single parameter which is the key of the value to be deleted.
[Map->Size]	Returns the number of elements (keys) in the map.
[Map->Values]	Returns an array of all the values specified in the map.

The following examples show how to manipulate a map by inserting, removing, and displaying elements. The examples are all based on the following array which contains the seven days of the week in English each with an integer key corresponding to their calendar order.

```
[Variable: 'DaysOfWeek' = (Map: 1='Sunday',
2='Monday',
3='Tuesday',
4='Wednesday',
5='Thursday',
6='Friday',
7='Saturday')]
```

To get values from a map:

- The value for a given key within the map can be retrieved using the [Map->Find] tag. The tag accepts a single parameter which is the key of the value to be returned. The key can be any value in Lasso. In the following example the numeric keys in the DaysOfWeek variable are used to return several days of the week.

```
[Output: $DaysOfWeek->(Find: 2)] → Monday
[Output: $DaysOfWeek->(Find: 4)] → Wednesday
[Output: $DaysOfWeek->(Find: 6)] → Friday
```

- All of the keys used within a map can be displayed using the [Map->Keys] tag. In the following example, the integer keys of the DaysOfWeek map are displayed.

```
[Output $DaysOfWeek->Keys]
```

```
→ (Array: (1), (2), (3), (4), (5), (6), (7))
```

- All of the values used within a map can be displayed using the [Map->Values] tag. In the following example, the string values of the DaysOfWeek map are displayed.

```
[Output $DaysOfWeek->Values]
```

```
→ (Array: (Sunday), (Monday), (Tuesday), (Wednesday), (Thursday), (Friday),
(Saturday))
```

- All of the elements in a map can be displayed using the [Iterate] ... [/Iterate] tags. In the following example, a temporary variable TempElement is set to the value of each element of the map in turn. The [Pair->First] and [Pair->Second] parts of each element are displayed.

```
[Iterate: $DaysOfWeek, (Variable: 'TempElement')]
<br>[Output: $TempElement->First] = [Output: $TempElement->Second]
[/Iterate]
```

```
→ <br>1 = Sunday
   <br>2 = Monday
   <br>3 = Tuesday
   <br>4 = Wednesday
   <br>5 = Thursday
   <br>6 = Friday
   <br>7 = Saturday
```

- Alternately, all of the elements in a map can be displayed using the [Loop] ... [/Loop] tags. In the following example, the [Map->Size] tag is used to return the size of the map and the [Map->Get] tag is used to return a particular element of the map. These tags function exactly like the same tags used on a pair array. A temporary variable TempElement is used to make the code easier to read.

```
[Loop: ($DaysOfWeek->Size)]
[Variable: 'TempElement' = ($DaysOfWeek->(Get: (Loop_Count)))]
<br>[Output: $TempElement->First] = [Output: $TempElement->Second]
[/Loop]
```

```

→ <br>1 = Sunday
   <br>2 = Monday
   <br>3 = Tuesday
   <br>4 = Wednesday
   <br>5 = Thursday
   <br>6 = Friday
   <br>7 = Saturday

```

Note: Map elements cannot be set using the [Map->Get] member tag. Instead, map elements should be inserted using the [Map->Insert] member tag with the same key value as an element in the map.

To insert values into a map:

Elements can be added to the map or the value for a given key can be changed within a map using the [Map->Insert] tag.

- Use the [Map->Insert] tag with a name/value parameter to insert a new value into a map. The following example shows how to add an Extra Saturday to the map stored in DaysOfWeek. No value is returned by the [Map->Insert] tag, but the new value for key 8 is retrieved using [Map->Find] to show that the new element has been added.

```

<?LassoScript
  $DaysOfWeek->(Insert: 8='Extra Saturday');
  Output: $DaysOfWeek->(Find: 8);
?>

```

→ Extra Saturday

- Use the [Map->Insert] tag with the name of a value already stored in the map to replace that value within the map. The following example shows how to change the value for key 8 to Extra Sabado, substituting the Spanish word for Saturday. No value is returned by the [Map->Insert] tag, but the new value for key 8 is retrieved using [Map->Find] to show that the element has been modified.

```

<?LassoScript
  $DaysOfWeek->(Insert: 8='Extra Sabado');
  Output: $DaysOfWeek->(Find: 8);
?>

```

→ Extra Sabado

To remove values from a map:

The value for a key can be removed from a map using the [Map->Remove] tag. The tag accepts a single parameter, the name of the element to be removed. In the following example, the Extra Sabado entry is removed from the map stored in DaysOfWeek.

```
<?LassoScript
  $DaysOfWeek->(Remove: 8);
?>
```

To display the elements of a map:

For debugging purposes all of the elements of a map can be output simply by displaying the value of the variable holding the map. This is a quick way to see the value stored in a map, but is not intended to be used to show to site visitors.

```
[Variable: 'DaysOfWeek']
```

```
→ (Map: (1)=(Sunday),
    (2)=(Monday),
    (3)=(Tuesday),
    (4)=(Wednesday),
    (5)=(Thursday),
    (6)=(Friday),
    (7)=(Saturday))
```

Maps vs Pair Arrays

Maps and pair arrays can both be used to store data which is retrieved by name. Maps store a single value per name. Pair arrays can store many different values for each name. Maps do not maintain the order of elements contained within them. Pair arrays do maintain the order of elements, though they generally cannot be sorted. Maps contain only values associated with names (although the names and values can be of any data type). Arrays can contain a combination of pairs and other data types.

Maps should be used when the set of keys by which data will be retrieved is unique. Maps can be used as an equivalent for database records. Maps provide fast lookup of a value associated with a key.

Pair arrays should be used when multiple values need to be stored with each key or when the order of elements stored in the array is important. Pair arrays are used to return name/value parameters from Lasso actions or within custom tags.

Pairs

A pair is a compound data type that stores two elements. Pairs are most commonly used when working with lists of command tags and name/

value parameters in concert with the [Action_Params] tag or when parsing parameters of a custom tag using the [Params] tag.

Table 6: Pair Tag

Tag	Description
[Pair]	Creates a pair with the specified name and value as the first and second elements.

To create a pair:

- The following example shows how to create a pair using the [Pair] tag. The tag accepts a single name/value parameter. The name part of the parameter becomes the First part of the pair. The value part of the parameter becomes the Second part of the pair.

```
[Pair: 'First_Name'='John']
```

- Pairs can be created in an [Array] constructor tag by specifying a name/value parameter as one of the parameters for the new array. The following example shows how to create an array with three pair elements.

```
[Array: 'First_Name'='John', 'Last_Name'='Doe', 'Phone_Number'='800-555-1212']
```

- Pairs are also created in a [Map] constructor tag by specifying name/value parameters. The following example shows how to create a map with three pairs.

```
[Map: 'First_Name'='John', 'Last_Name'='Doe', 'Phone_Number'='800-555-1212']
```

Pair Member Tags

The pair data type has two member tags that can be used to change or retrieve the two elements of the pair data type.

Table 7: Pair Member Tags

Tag	Description
[Pair->First]	Returns the first element of the pair. Can be used as the left parameter of an assignment operator to change the first element of the pair.
[Pair->Second]	Returns the second element of the pair. Can be used as the left parameter of an assignment operator to change the second element of the pair.

Note: For compatibility with maps and arrays the `[Pair->Size]` tag always returns 2 and `[Pair->(Get:1)]` and `[Pair->(Get:2)]` work to extract the first and second elements from a pair.

To get the elements of a pair:

The following example shows how to create a pair using a name/value parameter and then return the First and Second elements of the pair.

```
[Variable: 'Test_Pair' = (Pair: 'First_Name'='John')]
[Output: $Test_Pair->First]: [Output: $Test_Pair->Second]
```

→ First_Name: John

To set the elements of a pair:

The following example shows how to set the first and second elements of a pair to new values using the assignment operator `=`. The altered pair is then displayed.

```
<?LassoScript
  Variable: 'Test_Pair' = (Pair: 'First_Name'='John');
  $Test_Pair->First = 'Last_Name';
  $Test_Pair->Second = 'Doe';
  Output: $Test_Pair->First + ' : ' + $Test_Pair->Second;
```

→ Last_Name: Doe

To display the elements of a pair:

For debugging purposes the elements of a pair can be displayed simply by outputting the variable which contains the pair. The following example shows how to output a pair stored in a variable `Test_Pair`.

```
[Variable: 'Test_Pair' = (Pair: 'First_Name'='John')]
[Variable: 'Test_Pair']
```

→ (Pair: (First_Name)=(John))

Common Maps and Arrays

Lasso returns values in maps or arrays from many different tags. The tags that return compound data structures are detailed in *Table 8: Common Maps and Arrays*. These tags are each covered in more detail in the appropriate chapter in this Lasso 7 Language Guide or in the Extending Lasso Guide.

Table 8: Common Maps and Arrays

Tag	Description
[Action_Params]	Returns an array of pairs representing the current Lasso action. Includes both command tags and name/value parameters.
[File_Uploads]	Returns an array of maps representing the files which were uploaded. The map contains information about each file such as its original name, temporary storage location, size, etc.
[Locals]	Returns a map which contains an element for each local variable defined in a custom tag.
[Params]	Returns an array of all the parameters passed to a custom tag. The array will contain single values or pairs depending on what parameters were passed.
[String->Split]	Returns an array with elements created by splitting the string at the character specified as a parameter.
[Tags]	Returns a map with each tag defined in LDML.
[Variables]	Returns a map which contains an element for each variable defined in the current page.

18

Chapter 18

Encoding

Lasso can be used to publish data in many different formats. Encoding ensures that only legal characters are used for the desired output format.

- *Overview* describes the different formats which LDML encoding supports.
- *Encoding Keywords* describes how to use encoding keywords to modify the output of substitution tags.
- *Encoding Controls* describes how to use the [Encode_Set] ... [/Encode_Set] tags to modify the default encoding for substitution tags.
- *Encoding Tags* describes the individual substitution tags which can be used to encode values.
- *Encryption Tags* describes tags for securely storing and transmitting data.
- *Compression Tags* describes tags for compressing string data for more efficient storage or transmission.

Overview

Encoding controls in LDML allow the developer to specify the format in which data output from substitution tags should be rendered. Encoding controls ensure that reserved or illegal characters are changed to entities so that they will display properly in the desired output format. Encoding controls allow for data to be output in any of the ways described in the *Encoding Formats* section below.

Encoding Rules

Encoding controls apply to the data output from tags differently depending on how the tags are used. Substitution tags have default HTML encoding if they output a value to a page. The value output from a nested substitution tag is not encoded. Substitution tags which contribute to the output of a LassoScript have default HTML encoding.

- **Substitution Tags** which output a value to the site visitor have a default encoding of `-EncodeHTML`. These tags are usually enclosed in square brackets and do not include nested tags which return values.

The default encoding ensures that any reserved or illegal characters in HTML are converted to HTML entities so they display properly. The default encoding can be overridden by explicitly including an encoding keyword in the substitution tag or using the `[Encode_Set] ... [/Encode_Set]` tags described below.

In the following example, some HTML code is output using the `[Output]` substitution tag. By default the angle brackets in the code are converted to HTML entities so they will display as angle brackets within a Web browser. If the `-EncodeNone` keyword is specified in the `[Output]` substitution tag then the angle brackets remain as text angle brackets and the HTML code will render as **Bold Text** within the Web browser.

[Output: 'Bold Text'] → Bold Text

[Output: 'Bold Text', -EncodeNone] → Bold Text

- **Nested Substitution Tags** are not encoded by default. This ensures that string calculations can be performed without having to specify any encoding keywords. However, the encoding of a nested substitution tag can be changed by explicitly including an encoding keyword. Care should be taken so that values are not encoded multiple times.

In the following example a string is stored in a variable using explicit HTML encoding. When the variable is output using the `-EncodeNone` tag, the value is output to the page with HTML encoding intact.

[Variable: 'HTML_Text' = (Output: 'Bold Text', -EncodeHTML)]

[Variable: 'HTML_Text', -EncodeNone] → Bold Text

- Tags within **LassoScripts** are encoded using the same rules for substitution tags. Tags which add to the output of the LassoScript are HTML encoded by default unless an explicit encoding keyword is specified or the `[Encode_Set] ... [/Encode_Set]` tags are used. Tags which are nested are not encoded by default unless an explicit encoding keyword is specified.

The following example shows a value output from a LassoScript first with the default HTML encoding, then with an explicit `-EncodeNone` keyword specified.

```
<?LassoScript
  Output: '<b>Bold Text</b>';
?>
```

→ `Bold Text`

```
<?LassoScript
  Output: '<b>Bold Text</b>', -EncodeNone;
?>
```

→ `Bold Text`

- **Square Bracketed Expressions** other than tags are not encoded by default. The use of the `[Output]` tag or one of the `[Encode_...]` tags is recommended to ensure that encoding is properly applied to string expressions. In the following example a string expression is output directly.

```
['<b>' + 'Bold Text' + '</b>']
```

→ `Bold Text`

Encoding Formats

The encoding controls in LDML can be used to output data in any of the following formats.

- **HTML Encoding** is the default output format. Reserved characters in HTML including `<` `>` `"` `&` are encoded into HTML entities. Extended-ASCII and foreign language characters are encoded into a numerical HTML entity for the character `&#nnn`. Use the `-EncodeHTML` keyword or the `[Encode_HTML]` substitution tag.
- **Smart HTML Encoding** encodes only extended-ASCII and foreign language characters. The reserved characters in HTML are not encoded. This allows HTML code to be displayed with the HTML markup intact and any unsafe characters encoded using HTML entities. Use the `-EncodeSmart` keyword or the `[Encode_Smart]` substitution tag.
- **Break Encoding** encodes carriage returns and line feeds within the text to HTML `
` tags. The remainder of the text is HTML encoded. Text can be formatted using the `-EncodeBreak` keyword or the `[Encode_Break]` substitution tag.
- **XML Encoding** encodes reserved characters such as `&` `'` `"` `<` `>` which are used to create the markup of XML into XML entities. This ensures that text used in XML tag names or attributes does not contain any reserved

characters. Use the `-EncodeXML` keyword or the `[Encode_XML]` substitution tag.

- **Simple URL Encoding** only encodes illegal characters such as `<>#%{'`"|\\^~[]@®` into URL entities specified as `%nn`. Simple URL encoding can be used to encode an entire URL without disturbing the basic structure of the URL. Use the `-EncodeURL` keyword or the `[Encode_URL]` substitution tag. The following example shows a URL encoded with the `[Encode_URL]` tag.

```
[Encode_URL: 'http://www.example.com/Action.Lasso?The Name=A Value']
```

→ `http://www.example.com/Action.Lasso?The%20Name=A%20Value`

- **Strict URL Encoding** encodes both the illegal characters shown above and the reserved characters in URLs including `;/?:@=&`. Strict URL encoding should only be used on the names or values included as name/value parameters. Use the `-EncodeStrictURL` keyword or the `[Encode_StrictURL]` substitution tag. The following example shows only the name/value parameter of a URL encoded with the `[Encode_StrictURL]` tag.

```
http://www.example.com/Action.Lasso?
```

```
[Encode_StrictURL: 'The Name']=[Encode_StrictURL: 'A Value']
```

→ `http://www.example.com/Action.Lasso?The%20Name=A%20Value`

- **SQL Encoding** changes any illegal characters in SQL string values into their escaped equivalents. Quote marks and backslashes are escaped so they don't interfere with the structure of the SQL statement.

```
[Encode_SQL: 'A "String" is born.']
```

→ `A \"String\" is born.`

- **Base 64 Encoding** changes any string value into a string of ASCII characters which can be safely transmitted through URLs or email. This algorithm is sometimes used to obscure data so it is difficult to read by a casual passerby without providing any actual security. Base64 is also used to transmit passwords (essentially as plain-text) to some Web servers.

Deactivate encoding for a substitution tag using the `-EncodeNone` keyword. By default, nested substitution tags will not have encoding applied so the `-EncodeNone` keyword is not required within nested substitution tags.

Encoding Keywords

Encoding keywords can be used within any substitution tag to modify the encoding of the output value of that tag. Substitution tags which output values to the page default to `-EncodeHTML` so this keyword does not need to be specified if HTML encoding is desired. Nested substitution tags are not encoded by default, specifying `-EncodeNone` in nested substitution tag is unnecessary.

Only one encoding keyword can be used in a tag. If multiple encodings are desired the `[Encode_...]` tags should be used.

Table 1: Encoding Keywords

Keyword	Description
-EncodeBreak	Encodes carriage returns and new line characters into HTML <code>
</code> breaks. The remainder of the text is HTML encoded.
-EncodeHTML	Encodes HTML reserved and illegal characters into HTML entities for highest fidelity display.
-EncodeNone	Performs no encoding.
-EncodeSmart	Encodes HTML illegal characters into HTML entities. Useful for encoding strings that contain HTML markup.
-EncodeStrictURL	Encodes all URL reserved and illegal characters into URL entities for highest fidelity data transmission.
-EncodeURL	Encodes URL illegal characters into URL entities. Useful for encoding entire URLs.
-EncodeXML	Encodes XML reserved and illegal characters into XML entities for highest fidelity data transmission.

Please consult the previous section *Encoding Formats* for information about what characters each encoding keyword modifies.

Using the encoding keywords:

The following example shows how text is output from the `[Output]` tag using first the default `-EncodeHTML` encoding and then an explicit `-EncodeNone` encoding.

```
[Output: '<b>Bold Text</b>'] → &lt;b&gt;Bold Text&lt;/b&gt;
[Output: '<b>Bold Text</b>', -EncodeNone] → <b>Bold Text</b>
```

Encoding Controls

The default encoding keyword for substitution tags which output values to the Web page being constructed can be modified using the [Encode_Set] ... [/Encode_Set] tags. All square bracketed substitution tags or tags within a LassoScript that output a value will use the encoding specified in surrounding [Encode_Set] ... [/Encode_Set] tags rather than the default HTML encoding.

The [Encode_Set] tag accepts a single parameter, an encoding keyword. Any of the valid encoding keywords from *Table 1: Encoding Keywords* can be used. All substitution tags which output values will behave as if this encoding keyword were specified within the tag.

Nested substitution tags (sub-tags) will not be affected by the [Encode_Set] ... [/Encode_Set] tags. Values from nested substitution tags are not encoded unless an encoding keyword is specified explicitly within each tag.

Table 2: Encoding Controls

Keyword	Description
[Encode_Set] ... [/Encode_Set]	Sets the default encoding for all substitution tags which output values within the container tag.

To change the default encoding for a LassoScript:

Start and end the LassoScript with [Encode_Set] ... [/Encode_Set] tags. In the following LassoScript HTML code is output using [Ouput] tags. The default encoding for all tags is set to -EncodeNone so that the HTML is rendered properly in the output.

```
<?LassoScript
  Encode_Set: -EncodeNone;
  Output: '<b>HTML Text</b>';
  /Encode_Set;
?>

→ <b>Bold Text</b>
```

Encoding Tags

The encoding substitution tags can be used to explicitly encode any string value. The output of these tags is the same as the output which would be produced by using the appropriate encoding keyword on a substitution tag that returned the same value.

Note: The encoding tags do not accept encoding keywords. Use nested encoding tags to perform multiple encodings.

Table 3: Encoding Tags

Keyword	Description
[Decode_Base64]	Decodes a string which has been encoded using the base 64 algorithm. Accepts one parameter, a string to be decoded.
[Decode_HTML]	Decodes HTML by changing HTML entities back into extended ASCII characters.
[Decode_URL]	Decodes a URL by changing URL entities back into extended ASCII characters.
[Encode_Base64]	Encodes a string using the base 64 algorithm. Accepts one parameter, a string to be encoded.
[Encode_Break]	Encodes carriage returns and new line characters into HTML breaks. The remainder of the text is HTML encoded.
[Encode_HTML]	Encodes HTML reserved and illegal characters into HTML entities for highest fidelity display.
[Encode_Smart]	Encodes HTML illegal characters into HTML entities. Useful for encoding strings that contain HTML markup.
[Encode_SQL]	Encodes illegal characters in SQL string literals by escaping them with a backslash.
[Encode_StrictURL]	Encodes all URL reserved and illegal characters into URL entities for highest fidelity data transmission.
[Encode_URL]	Encodes URL illegal characters into URL entities. Useful for encoding entire URLs.
[Encode_XML]	Encodes XML reserved and illegal characters into XML entities for highest fidelity data transmission.

Using the encoding tags:

The following example shows how text is output from the [Encode_HTML] tag with all HTML reserved characters encoded. The same text is then output from an [Output] tag with an encoding keyword of -EncodeNone specified.

[Encode_HTML: 'Bold Text'] → Bold Text
[Output: 'Bold Text', -EncodeNone] → Bold Text

Encryption Tags

LDML provides a number of tags which allow data to be encrypted for secure storage or transmission. Three different types of encryption are supplied.

- **BlowFish** is a fast, popular encryption algorithm. Lasso provides tools to encrypt and decrypt string values using a developer-defined seed. This is the best tag to use for data which needs to be stored in a database or transmitted securely.
- **MD5** is a one-way encryption algorithm that is often used for passwords. There is no way to decrypt data which has been encrypted using MD5. See below for an example of how to use MD5 to store and check passwords securely.

Table 4: Encryption Tags

Tag	Description
[Decrypt_BlowFish]	Decrypts a string. Accepts two parameters, a string to be decrypted and a -Seed keyword with the key or password for the decryption.
[Encrypt_BlowFish]	Encrypts a string. Accepts two parameters, a string to be encrypted and a -Seed keyword with the key or password for the encryption.
[Encrypt_MD5]	Encrypts a string. Accepts one parameter, a string to be encrypted. Returns a fixed size hash value for the string which cannot be decrypted.

Note: The BlowFish tags are not binary safe. The output of the tag will be truncated after the first null character. It is necessary to use [Encode_Base64] or [Encode_UTF8] prior to encrypting data that might contain binary characters using these tags.

BlowFish Seeds

BlowFish requires a seed in order to encrypt or decrypt a string. The same seed which was used to encrypt data using the [Encrypt_BlowFish] tag must be passed to the [Decrypt_BlowFish] tag to decrypt that data. If you lose the key used to encrypt data then the data will be essentially unrecoverable.

Seeds can be any string between 4 characters and 112 characters long. Pick the longest string possible to ensure a secure encryption. Ideal seeds contain a mix of letters, digits, and punctuation.

The security considerations of storing, transmitting, and hard coding seed values is beyond the scope of this manual. In the examples that follow, we present methodologies which are easy to use, but may not provide the highest level of security possible. You should consult a security expert if security is very important for your Web site.

Note: The BlowFish algorithm will return random results if you attempt to decrypt data which is not actually encrypted.

To store data securely in a database:

Use the [Encrypt_BlowFish] and [Decrypt_BlowFish] tags to encrypt data which will be stored in a database and then to decrypt the data when it is retrieved from the database.

- 1 Store the data to be encrypted into a string variable, PlainText.

```
[Variable: 'PlainText' = 'The data to be encrypted.']
```

- 2 Encrypt the data using the [Encrypt_BlowFish] tag with a hard-coded -Seed value. Store the result in the variable CypherText.

```
[Variable: 'CypherText' = (Encrypt_BlowFish: (Variable: 'PlainText'),  
-Seed='This is the blowfish seed')]
```

- 3 Store the data in CypherText in the database. The data will not be viewable without the seed. The following [Inline] ... [/Inline] creates a new record in an Contacts database for John Doe with the CypherText.

```
[Inline: -Add,  
-Database='Contacts',  
-Table='People',  
-KeyField='ID',  
'First_Name'='John',  
'Last_Name'='Doe',  
'CypherText'=(Variable: 'CypherText')]  
[/Inline]
```

- 4 Retrieve the data from the database. The following [Inline] ... [/Inline] fetches the record from the database for John Doe and places the CypherText into a variable named CypherText.

```
[Inline: -Search,  
-Database='Contacts',  
-Table='People',  
-KeyField='ID',
```

```

        'First_Name'='John',
        'Last_Name'='Doe']
    [Variable: 'CypherText' = (Field: 'CypherText')]
[/Inline]

```

- 5 Decrypt the data using the [Decrypt_BlowFish] tag with the same hard-coded -Seed value. Store the result in the variable PlainText.

```

    [Variable: 'PlainText' = (Decrypt_BlowFish: (Variable: 'CypherText'),
        -Seed='This is the blowfish seed')]

```

- 6 Display the new value stored in PlainText.

```

    [Variable: 'PlainText']

```

→ The data to be encrypted.

To store and check encrypted passwords:

The [Encrypt_MD5] tag can be used to store a secure version of a password for a site visitor. On every subsequent visit, the password given by the visitor is encrypted using the same tag and compared to the stored value. If they match, then the visitor has supplied the same password they initially supplied.

- 1 When the visitor creates an account use [Encrypt_MD5] to create an encrypted version—a fixed size hash value—of the password they supply. In the following example, the password they supply is stored in the variable VisitorPassword and the encrypted version is stored in SecurePassword.

```

    [Variable: 'SecurePassword' = (Encrypt_MD5: (Variable: 'VisitorPassword'))]

```

- 2 Store this MD5 hash value for the password in a database along with the visitor's username.
- 3 On the next visit, prompt the visitor for their username and password. Fetch the record identified by the visitor's specified username and retrieve the MD5 hash value stored in the field SecurePassword.
- 4 Use [Encrypt_MD5] to encrypt the password that the visitor has supplied and compare the result to the stored, encrypted MD5 hash value that was generated from the password they supplied when they created their account.

```

    [If: (Encrypt_MD5: (Variable: 'VisitorPassword')) == (Field: 'SecurePassword')]
        Log in successful.
    [Else]
        Password does not match.
    [/If]

```

Note: For more security, most log-in solutions require both a username and a password. The password is not checked unless the username matches first. This prevents site visitors from guessing passwords unless they know a valid

username. Also, many log-in solutions restrict the number of login attempts that they will accept from a client's IP address.

Compression Tags

LDML provides two tags which allow data to be stored or transmitted more efficiently. The [Compress] tag can be used to compress any text string into an efficient byte stream that can be stored in a text field in a database or transmitted to another server. The [Decompress] tag can then be used to restore a compressed byte stream into the original string.

The compression algorithm should only be used on large string values. For strings of less than one hundred characters the algorithm may actually result in a larger string than the source.

These tags can be used in concert with the [Null->Serialize] tag that creates a string representation of any data type in LDML and the [Null->UnSerialize] tag that returns the original value based on a string representation. An example below shows how to compress and decompress an array variable.

Table 5: Compression Tags

Tag	Description
[Compress]	Compresses a string parameter.
[Decompress]	Decompresses a byte stream.

To compress and decompress a string:

- 1 Use the [Compress] tag on the variable InputVariable holding the string value you want to compress. The result is a byte stream that represents the string which is stored in CompressedVariable.

```
[Variable: 'InputVariable'='This is the string to be compressed.']
[Variable: 'CompressedVariable'=(Compress: $InputVariable)]
```

- 2 The CompressedVariable can now be decompressed using the [Decompress] tag. The result is stored in OutputVariable and finally displayed.

```
[Variable: 'OutputVariable'=(Decompress: $CompressedVariable)]
[Variable: 'OutputVariable']
```

→ This is the string to be compressed.

To compress and decompress an array:

- 1 Store the array in a variable ArrayVariable.

```
[Variable: 'ArrayVariable'=(Array: 'one', 'two', 'three', 'four', 'five')]
```

- 2 Use the [Null->Serialize] tag to change the array into a string stored in InputVariable.
`[Variable: 'InputVariable'=$ArrayVariable->Serialize]`
- 3 Use the [Compress] tag on the variable InputVariable holding the string representation for the array. The result is a byte stream which is stored in CompressedVariable.
`[Variable: 'CompressedVariable'=(Compress: $InputVariable)]`
- 4 The CompressedVariable can now be decompressed using the [Decompress] tag. The result is a string stored in OutputVariable.
`[Variable: 'OutputVariable'=(Decompress: $CompressedVariable)]`
- 5 The string representation of the array can now be changed back into the array by creating a new variable ArrayVariable and then calling the [Null->UnSerialize] tag with OutputVariable as a parameter.
`[Variable: 'ArrayVariable'=Null]
[$ArrayVariable->(UnSerialize: $OutputVariable)]`
- 6 Finally, the original array can be output.
`[Variable: 'ArrayVariable']`
→ (Array: (one), (two), (three), (four), (five))

19

Chapter 19

Sessions

This chapter documents sessions and server-side variables.

- *Overview* describes how sessions operate and how sessions can be used.
- *Session Tags* describes the tags which can be used to create, manipulate, and delete sessions.
- *Session Example* describes how to use sessions to store site preferences.

Overview

Sessions allow variables to be created which are persistent from page to page within a Web site. Rather than passing data from page to page using HTML forms or URLs, data can be stored in ordinary LDML variables which are automatically stored and retrieved by Lasso on each page a visitor loads.

Sessions are very easy to use, but the intricacies can be rather difficult to explain. The *Session Examples* section later in this chapter presents three examples for how to use sessions to perform common tasks. These examples should be consulted first to see real world examples of sessions in action before reading through the tag reference sections.

Ways in which sessions can be used:

- **Current State** – Sessions can store the current state of a Web site for a given visitor. They can determine what the last search they performed was, how the data on a results page was sorted, or in what format the data should be presented.
- **Store References to Database Records** – Key field values can be stored in a session for quick access to records associated with a site

visitor. These might include records in a user database or shopping cart database.

- **Store Authentication Information** – After a visitor has authenticated themselves using a username and password, that authentication information can be stored in a session and then checked on each page to ensure that the same visitor is accessing data from page to page.
- **Store Data Without Using a Database** – Complex data types such as arrays and maps can be stored in session variables. In a Web site with multiple forms the data from each form can be stored in a session and only placed in the database once the final form is submitted. Or, a shopping cart can be stored in a session and only placed in an orders database on checkout.

How Sessions Work

A session has three characteristics: a name, a list of variables that should be stored, and an ID string that identifies a particular site visitor.

- **Name** – The session name is defined when the session is created by the [Session_Start] tag. The same session name must be used on each page in the site which wants to load the session. The name usually represents what type of data is being stored in the session, e.g. Shopping_Cart or Site_Preferences.
- **Variables** – Each session maintains a list of variables which are being stored. Variables can be added to the session using [Session_AddVariable]. The values for all variables in the session are remembered at the bottom of each page which loads the session. The last value for each variable is restored when the session is next loaded.
- **ID** – Lasso automatically creates an ID string for each site visitor when a session is created. The ID string is either stored in a cookie or passed from page to page using the -Session command tag. When a session is loaded the ID of the current visitor is combined with the name of the session to load the particular set of variables for the current visitor.

Sessions are created and loaded using the [Session_Start] tag. This tag should be used on the top of each page which needs access to the shared variables. The [Session_Start] either creates a new session or loads an existing session depending on what session name is specified and the ID for the current visitor.

Sessions can be set to expire after a specified amount of idle time. The default is 15 minutes. If the visitor has not loaded a page which starts the session within the idle time then the session will be deleted automatically. Note that the idle timeout restarts every time a page is loaded which starts the session.

Once a variable has been added to a session using the [Session_AddVariable] tag it will be set to its stored value each time the [Session_Start] tag is called. The variable does not need to be added to the session on each page. A variable can be removed from a session using the [Session_RemoveVariable] tag. This tag does not alter the variable’s value on the current page, but prevents the value of the variable from being stored in the session at the end of the current page.

Session Tags

Each of the session tags is described in *Table 1: Session Tags*. The parameters for [Session_Start] are described in more detail in *Table 2: [Session_Start] Parameters*.

Table 1: Session Tags

Tag	Description
[Session_Start]	Starts a new session or loads an exisiting session. Accepts four parameters: -Name is the name of the session to be started. Additional parameters are described in Table 2: [Session_Start] Parameters.
[Session_ID]	Returns the current session ID. Accepts a single parameter: -Name is the name of the session for which the session ID should be returned.
[Session_AddVariable]	Adds a variable to a specified session. Accepts two parameters: -Name is the name of the session and a second unnamed parameter is the name of the variable.
[Session_RemoveVariable]	Removes a variable from a specified session. Accepts two parameters: -Name is the name of the session and a second unnamed parameter is the name of the variable.
[Session_End]	Deletes the stored information about a named session for the current visitor. Accepts a single parameter: - Name is the name of the session to be deleted.
[Session_Abort]	Prevents the session from being stored at the end of the current page. This allow graceful recovery from an error that would otherwise corrupt data stored in the session.
[Session_Result]	When called immediately after the [Session_Start] tag, returns "new", "load", or "expire" depending on whether a new session was created, an existing session loaded, or an expired session forced a new session to be created.

Table 2: [Session_Start] Parameters

Keyword	Description
-Name	The name of the session.
-Expires	The idle expiration time for the session in minutes.
-ID	The ID for the current visitor. If no ID is specified then the cookie and link parameters will be inspected for valid visitor IDs.
-UseCookie	If specified then site visitors will be tracked by cookie. -UseCookie is the default unless -UseLink or -UseNone is specified.
-UseLink	If specified then site visitors will be tracked by modifying all the absolute and relative links in the current format file.
-UseNone	No links on the current page will be modified and a cookie will not be set. -UseNone allows custom session tracking to be used.

Note: -UseCookie is the default for [Session_Start] unless -UseLink is or -UseNone is specified. Use -UseLink to track a session using only links. Use both -UseLink and -UseCookie to track a session using both links and a cookie.

Starting a Session

The [Session_Start] tag is used to start a new session or to load an existing session. When the [Session_Start] tag is called with a given -Name parameter it first checks to see whether an ID is defined for the current visitor. The ID is searched for in the following three locations:

- **ID** – If the [Session_Start] tag has an -ID parameter then it is used as the ID for the current visitor.
- **Cookie** – If a session tracker cookie is found for the name of the session then the ID stored in the cookie is used.
- **Session** – If a -Session command tag for the name of the session was specified in the link that loaded the current page then the parameter of that tag is used as the session ID.

The name of the session and the ID are used to check whether a session has already been created for the current visitor. If it has then the variables in the session are loaded replacing the values for any variables of the same name that are defined on the current page.

If no ID can be found, the specified ID is invalid, or if the session identified by the name and ID has expired then a new session is created.

After the [Session_Start] tag has been called the [Session_ID] tag can be used to retrieve the ID of the current session. It is guaranteed that either a valid session will be loaded or a new session will be created by the [Session_Start] tag.

Note: The [Session_Start] tag must be used on each page you want to access session variables.

Session Tracking

The session ID for the current visitor can be tracked using two different methods or a custom tracking system can be devised. The tracking system depends on what parameters are specified for the [Session_Start] tag.

- **Cookie** – The default session tracking method is using a cookie. If no other method is specified when creating a session then the -UseCookie method is used by default. The cookie will be inspected automatically when the visitor loads another page in the site which includes a [Session_Start] tag. No additional programming is required.

The session tracking cookie is of the following form. The name of the cookie includes the words _Session_Tracker_ followed by the name given to the session in [Session_Start]. The value for the cookie is the session ID as returned by [Session_ID].

```
_SessionTracker_SessionName=1234567890abcdefg
```

- **Links** – If the -UseLink parameter is specified in the [Session_Start] tag then Lasso will automatically modify links contained on the current page. The preferences for which links will be modified by Lasso can be adjusted in the *Setup > Global Settings > Sessions* section of Lasso Administration. See the Lasso Professional 7 Setup Guide for more information. No additional programming beyond specifying the -UseLink parameter is required.

By default, links contained in the href parameter of ... and in the action parameter of <form action="..."> ... </form> tags will be modified.

Links are only modified if they reference a file on the same machine as the current Web site. Any links which start with any of the following strings are not modified.

file://	ftp://	http://	https://
javascript:	mailto:	telnet://	#

Links are modified by adding a -Session command tag to the end of the link parameters. The value of the command tag is the session name followed by a colon and the session ID as returned by [Session_ID]. For

example, an anchor tag referencing the current file would appear as follows after the `-Session` tag was added.

```
<a href="default.lasso?-Session=SessionName:1234567890abcdefg"> ... </a>
```

To start a session:

A session can be started using the `[Session_Start]` tag. The optional `-Expires` parameter specifies how long in minutes the session should be maintained after the last access by the site visitor. The default is 15 minutes. The optional `-UseLink` keyword specifies that absolute and relative links in the current format file should be modified to contain a reference to the session. The optional `-UseCookie` keyword specifies that a cookie should be set in the visitor's Web browser so that the session can be retrieved in subsequent pages.

The following example starts a session named `Site_Preferences` with an idle expiration of 24 hours (1440 minutes). The session will be tracked using both cookies and links.

```
[Session_Start: -Name='Site_Preferences', -Expires='1440', -UseLink, -UseCookie]
```

When the `[Session_Start]` tag is called it restores all stored variables. If a variable by the same name has already been created on the page then that variable value will be overwritten by the stored variable value.

To add variables to a session:

Use the `[Session_AddVariable]` tag to add a variable to a session. Once a variable has been added to a session its value will be remembered at the end of each format file in which the variable is used. Variables included in a session will be automatically defined when the `[Session_Start]` tag is called. In the following example a variable `RealName` is added to a session named `Site_Preferences`.

```
[Session_AddVariable: -Name='Site_Preferences', 'Real_Name']
```

Variables will not be created by the `[Session_AddVariable]` tag. Each `[Session_AddVariable]` should be accompanied by a `[Variable]` tag that defines the starting value for the variable.

To remove variables from a session:

Use the `[Session_RemoveVariable]` tag to remove a variable from a session. The variable will no longer be stored with the session and its value will not be restored in subsequent pages. The value of the variable in the current page will not be affected. In the following example a variable `RealName` is removed from a session named `Site_Preferences`.

```
[Session_RemoveVariable: -Name='Site_Preferences', 'Real_Name']
```

To delete a session:

A session can be deleted using the [Session_End] tag with the name of the session. The session will be ended immediately. None of the variables in the session will be affected in the current page, but their values will not be restored in subsequent pages. Sessions can also end automatically if the timeout specified by the -Expires keyword is reached. In the following example the session Site_Preferences is ended.

```
[Session_End: -Name='Site_Preferences']
```

To pass a session in an HTML form:

Sessions can be added to URLs automatically using the -UseLink keyword in the [Session_Start] tag. In order to pass a session using a form a hidden input must be added explicitly. The hidden input should have the name -Session and the value Session_Name:Session_ID. In the following example, the ID for a session Site_Preferences is returned using [Session_ID] and passed explicitly in an HTML form.

```
<form action="reponse.lasso" method="POST">
  <input type="hidden" name="-Session"
    value="Site_Preferences:[Session_ID: -Name='Site_Preferences']">
  ...
  <input type="submit" name="-Nothing" value="Submit Form">
</form>
```

To track a session using links only if cookies are disabled:

The following example shows how to start a session using links if cookies are disabled. The cookie which is created by the session tracker has the name _SessionTracker_SessionName. If this cookie does not exist then the user does not have cookies enabled. The following code checks for this cookie and specifies -UseLink and -UseCookie if it is not present or only -UseCookie if it is.

```
[Variable: 'Session_Name' = 'Site_Preferences']
[If: (Cookie_Value: '_SessionTracker_' + $Session_Name) == ""]
  [Session_Start: -Name=$Session_Name, -UseLink, -UseCookie]
[Else]
  [Session_Start: -Name=$Session_Name, -UseCookie]
[/If]
```

Session Example

This example demonstrates how to use sessions to store site preferences which are persistent from page to page.

Web sites can be customized for individual visitors using sessions. In this example a site visitor is allowed to enter certain information about themselves in various forms throughout the Web site. When subsequent forms are encountered, the Web site should be able to pre-fill any elements that the visitor has already specified.

Sessions will be used to track the visitors `RealName`, `EmailAddress`, and `FavoriteColor` in three variables.

To create the session:

The following code will be specified at the top of every Web page in the Web site. The session must be started in every Web page which requires access to or which might modify the stored variables.

- 1 The `[Session_Start]` tag is used to start a session named `Site_Preferences`. The expiration of the session is set to 24 hours (1440 minutes). The session will be tracked by both links and cookies.

```
[Session_Start: -Name='Site_Preferences', -Expires='1440', -UseLink, -UseCookie]
```

- 2 The three variables `RealName`, `EmailAddress`, and `FavoriteColor` are added to the session using `[Session_AddVariable]`.

```
[Session_AddVariable: -Name='Site_Preferences', 'RealName']
[Session_AddVariable: -Name='Site_Preferences', 'EmailAddress']
[Session_AddVariable: -Name='Site_Preferences', 'FavoriteColor']
```

- 3 Finally, default values are established for all three variables. `RealName` and `EmailAddress` are set to the empty string if they are not defined. `FavoriteColor` is set to blue `#0000cc` if it has not been defined. These default values will only be set the first time the session is started. In subsequent pages, the variables will automatically be set to the value stored in the session.

```
[If: (Variable_Defined: 'RealName' ) == False]
[Variable: 'RealName' = " ]
[/If]

[If: (Variable_Defined: 'EmailAddress') == False]
[Variable: 'EmailAddress' = " ]
[/If]

[If: (Variable_Defined: 'FavoriteColor') == False]
[Variable: 'FavoriteColor' = '#0000cc']
[/If]
```


To use the session variables:

The session variables are used in each page as normal variables. Whatever value they are set to at the end of the Web page will be the value the variable has the next time the session is started.

- The FavoriteColor variable can be used to set the color of text by using it in an HTML tag. In the following example, the visitors RealName will be shown in the specified color.

```
<font color="[Variable: 'FavoriteColor']"> Welcome [Variable: 'RealName'] </font>
```

- The visitor's RealName and EmailAddress can be shown in a form by placing the variables in the HTML <input> tags. The following form allows the visitor to enter their name and email address and to select a favorite color from a pop-up menu.

```
<form action="response.lasso" method="POST">
  <br>Your Name:
    <input type="text" name="RealName" value="[Variable: 'RealName']">
  <br>Your Email Address:
    <input type="text" name="EmailAddress" value="[Variable: 'EmailAddress']">
  <br>Your Favorite Color:
    <select name="FavoriteColor">
      <option value="#0000cc"> Blue </option>
      <option value="#cc0000"> Red </option>
      <option value="#009900"> Green </option>
    </select>
  <br>
  <input type="submit" name="-Nothing" value="Submit">
</form>
```

In the response page response.lasso, the form inputs can be retrieved using the [Action_Param] tag and stored into variables. These new values will now be stored with the session.

```
[Variable: 'RealName' = (Action_Param: 'RealName')]
[Variable: 'EmailAddress' = (Action_Param: 'EmailAddress')]
[Variable: 'FavoriteColor' = (Action_Param: 'FavoriteColor')]
```


20

Chapter 20

Files and Logging

LDML provides three sets of tags that create and manipulate files on the Web server: include tags, logging tags, and file tags.

- ***Includes*** describes how to include format files and library files within the current format file.
- ***Logging*** describes the [Log_...] tags which allow data to be written to a text file or to Lasso's internal error logs.
- ***File Tags*** describes the [File_...] tags which allow files and directories to be created, read, written, edited, moved, and deleted.
- ***File Uploads*** describes the [File_Uploads] tag which allows files that have been uploaded with an HTML form to be manipulated.
- ***File Streaming Tags*** describes the [File] and [Directory] tags and data types, and their various member tags that allow files and folders to be manipulated using an object-oriented methodology.

Includes

LDML allows format files to be included within the current format file. This can be very useful for setting up site-wide navigation elements (e.g. page headers and footers), separating the graphical elements of a site from the programming elements, and for organizing a project into reusable code components. There are three types of files that can be included with the various include tags depending on how the LDML code and other data in the included file needs to be treated.

- **Format Files** can be included using the [Include] tag. The LDML code within the included format file executes at the location of the [Include] tag

as if it were part of the current file. Any HTML code or text within the format file is inserted into the current format file.

```
[Include: 'format.lasso']
```

- **Text or Binary Data** can be included using the `[Include_Raw]` tag. No LDML code in the included file is processed and no encoding is performed on the included data.

```
[Include_Raw: 'Picture.gif']
```

- **LDML Code** can be included using the `[Library]` tag. No output is returned from the `[Library]` tag, but any LDML code within the file is executed.

```
[Library: 'library.lasso']
```

- **Variables** can be set to the contents of a file using the `[Include]` and `[Include_Raw]` tags. The `[Include]` tag inserts the results of processing any LDML code within the file into the variable. The `[Include_Raw]` tag inserts the raw text or binary data within the file into the variable.

```
[Variable: 'File_Data' = (Include: 'format.lasso')]
```

```
[Variable: 'File_Data' = (Include_Raw: 'Picture.gif')]
```

See *Chapter 26: Images and Multimedia* for tips about how to use `[Include,...]` tags to serve images and multimedia files from Lasso.

Library Files

Library files are format files which are used to modify Lasso's programming environment by defining new tags and data types, setting up global constants, or performing initialization code. Libraries can be included within a format file using the `[Library]` tag or can be added to the global environment by placing the library file within the `LassoStartup` folder and then restarting Lasso Service.

Specifying Paths

All included files reference paths relative to the format file which contains the include tag. The path specified to the file is usually the same as the relative or absolute path which would be specified within an HTML anchor tag to reference the same file.

Files in the same folder as the current format file can be included by specifying the name of the file directly. The following tag includes a file named `Format.lasso` in the same folder as the file this tag is specified within.

```
[Include: 'Format.lasso']
```

Files in sub-folders within the same folder as the current format file can be included by specifying the relative path to the file which is to be included. The following tag includes a library file named `Library.lasso` within a folder named `Includes` that is in the same folder as the file this tag is specified within.

```
[Library: 'Includes/Library.lasso']
```

Files in other folders within the Web serving folder should be specified using absolute paths from the root of the Web serving folder. The `../` construct cannot be used to navigate up through the hierarchy of folders. The following tag includes an image file called `Picture.gif` from the `Images` folder contained in the root of the Web serving folder.

```
[Include_Raw: '/Images/Picture.gif']
```

File Suffixes

Any file which is included by Lasso including format files, library files, and images or multimedia files must have an authorized file suffix within Lasso Administration. See *Chapter 6: Setting Global Preferences* of the Lasso Professional 7 Setup Guide for more information about how to authorize file suffixes.

By default the following suffixes are authorized within Lasso Administration. Any of these files suffixes can be used for included files. The `.inc` file suffix is often used to make clear the role of format files which are intended to be included.

<code>.htm</code>	<code>.html</code>
<code>.inc</code>	<code>.Lasso</code>
<code>.LassoApp</code>	<code>.text</code>
<code>.txt</code>	<code>.uld</code>
<code>.pdf</code>	<code>.xml</code>
<code>.gif</code>	<code>.jpg</code>
<code>.psd</code>	<code>.png</code>
<code>.bmp</code>	<code>.tif</code>
<code>.rgb</code>	<code>.cmyk</code>

Error Controls

`Includes` suppress many errors from propagating out to the including page. If a syntax error occurs in an included file then the `[Include]` tag will return the reported error to the site visitor. If a logical error occurs in an included file then the `[Include]` tag will return the contents of the error page with the error reported. Techniques for debugging included files are listed on the following pages.

Table 1: Include Tags

Tag	Description
[Include]	Inserts the specified format file into the current format file. Any LDML code in the included format file is executed. Accepts a single parameter, the path and name of the format file to be included.
[Include_Raw]	Inserts the specified file into the current format file. No processing or encoding is performed on the included file. Accepts a single parameter, the path and name of the file to be included.
[Library]	Executes any LDML code in the specified format file, but, inserts no result into the current format file. Accepts a single parameter, the path and name of the format file to be executed.

Note: See *Chapter 27: HTTP/HTML Content and Controls* for documentation of the [Include_URL] tag. Lasso Professional 7 also supports the [Include_CGI] tag from Lasso Web Data Engine 3.x, but its use has been deprecated. The [Include_CGI] tag may not be supported in a future version of Lasso.

To include a format file:

Use the include file with the path to the format file which is to be included. The included format file will be processed and the results will be inserted into the current format file as if the code had been specified within the current file at the location of the [Include] tag. The following example shows how to include a file named `format.lasso` which is contained in the same folder as the current format file.

```
[Include: 'format.lasso']
```

To include a library file:

Library files which contain custom tag definitions or LDML code that does not return any output can be included using the [Library] tag. The LDML code within the library file will be executed, but no result will be returned to the current format file. The following example shows how to include a library file named `library.lasso` which is contained in the same folder as the current format file.

```
[Library: 'library.lasso']
```

To debug an included file:

The include tags do return errors that occur in the included file, but it can be difficult to debug problems in included files. The errors from an included file can sometimes be more easily seen by loading the

file directly within a Web browser. This will reveal any syntax errors within the included file and ensure that all the code in the included file performs properly. The following URL references a format file named `format.lasso` inside an `Includes` folder.

`http://www.example.com/Includes/format.lasso`

Note: Some include files rely on variables from the format file that includes them to operate properly. These include files cannot be debugged by simply loading them in a Web browser.

To debug an included library file:

Since library files do not ordinarily return any output to the current format file they can be difficult to debug.

To debug an included library file, insert debugging messages within the code of the library file using the `[Output]` tag. Ordinarily, these messages will never be seen since the `[Library]` tag does not return any output. The following example shows an `[Output]` tag reporting the current error.

```
[Output: (Error_CurrentError: -ErrorCode) + ': ' + (Error_CurrentError)]
```

If the `[Library]` tag which includes the code library is changed to an `[Include]` tag then the output of the `[Output]` tag will be inserted into the current format file. This allows the debugging messages to be seen. Once the file is working successfully, the `[Include]` can be changed back to a `[Library]` tag to hide the debugging messages.

To prevent included files from being served directly:

Included files can be named with any file suffix which is authorized within Lasso Administration. If a file suffix is authorized within Lasso Administration, but is set to not be served by the Web server application then files with that file suffix can only be used as include files and can never be served directly. For example, to authorize the `.inc` file suffix the following steps must be taken.

- 1 Authorize `.inc` in Lasso Administration *Setup > Settings > File Extensions*.
- 2 Using the file suffix controls of your Web server applications, deny the suffix `.inc` so that files with that suffix cannot be served. This can usually be accomplished with specific file suffix controls or with a Web server realm. Consult the Web server documentation for more information.

Note: If LDML code is placed in an include file that is authorized for processing by Lasso (step 1 above), but is not set in the Web server preferences to always be processed by Lasso or never to be served (step 2 above),

then it may be possible for site visitors to view the unprocessed LDML code by loading the include file directly.

Advanced Methodology

Includes and library files allow LDML format files to be structured in order to create reusable components, separate programming logic from data presentation, and in general to make Web sites easier to maintain. There are many different methods of creating structured Web sites which are beyond the scope of this manual. Please consult the third party resources at the Blue World Web site for more information.

Logging

The [Log_...] tags allow information output by LDML format files to be logged to a specified text file, or to one of Lasso's internal error logs with a predefined error level. The [Log_...] tags can be used to keep track of what pages site visitors are visiting or what database actions they have performed. They can be used to write debugging information to Lasso Service's console window, or as a way to log specific format file errors to Lasso's internal error logs.

Logging to File

When executed, the contents of the [Log] ... [/Log] container tags is appended to a specified text file. The [Log] ... [/Log] tags can write to any text file provided that the file is on the same machine as Lasso Service. The [Log_...] tags cannot output to the format file that contains them. All returns, tabs, and spaces between the [Log] ... [/Log] tags will be included in the output data.

The following [Log] ... [/Log] tags output a single line containing the date and time with a return at the end to the file specified. The tags are shown first with a Windows path, then with a Mac OS X path.

```
[Log: 'C://Logs/LassoLog.txt'] [Server_Date] [Server_Time]
[/Log]

[Log: '///Logs/LassoLog.txt'] [Server_Date] [Server_Time]
[/Log]
```

The path to the directory where the log will be stored should be specified according to the same rules as those for the file tags. See the *File Tags* section of this chapter for full details about relative, absolute, and fully qualified paths on both Mac OS X and Windows.

Table 2: File Log Tags

Tag	Description
[Log] ... [/Log]	Logs the contents of the container tags to a specified text file. Requires the path to the text file as the only parameter:

To log site visits to a file:

Use the [Server_...] and [Client_...] tags to return information about the current visitor and what page they are visiting. The following code will log the current date and time, the visitor's IP address, the name of the server and the page they were loading, and the GET and POST parameters that were specified.

```
[Log: 'E://Logs/LassoLog.txt'] [Server_Date: -Extended] [Server_Time: -Extended]
[Client_IP] [Server_Name] [Response_FilePath] [Client_GETArgs] [Client_POSTArgs]
[/Log]
```

See *Chapter 27: HTTP/HTML Content and Controls* for more information about the [Client_...] and [Server_...] tags.

To automatically roll log files by date:

Include a date component in the name of the log file. Since the date component will change every day, a new log file will be created the first time an item is logged each day. [Server_Date: -Extended] creates a safe date format to use. The following example logs to a file named e.g. 2001-05-31.txt.

```
[Variable: 'Log_File' = ///Logs/' + (Server_Date: -Extended) + '.txt']
[Log: (Variable: 'Log_File')[Server_Date] [Server_Time]
[/Log]
```

Logging to Lasso's Internal Error Logs

In Lasso Professional 7, one has the option to log custom data to the Lasso internal error logs with a defined Lasso error level. Lasso's internal error logs include the following:

- The _Errors table in the Lasso_Internal Lasso MySQL database, viewable via the *Utility > Errors > Lasso Errors* page in Lasso Administration.
- The Lasso Service console window.
- The LassoErrors.txt file, located in the Lasso Professional 7 folder on the hard drive.

Lasso Professional 7 uses each of the error logs listed above to log its own internal errors, which are separate from user-defined logs. Every error in

Lasso’s internal error logs are flagged with an error level of Critical, Warning, or Detail, and which internal log that is used for each Lasso error level can be explicitly set in the *Monitor > Errors > Setup* section of Lasso Administration. For information on setting up Lasso’s internal error logs as well as descriptions of each error level, see *Chapter 9: Administration Utilities* in the Lasso Professional 7 Setup Guide.

The [Log_Critical], [Log_Warning], and [Log_Detail] tags are used to log custom data to the Lasso internal error logs with a defined Lasso error level of Critical, Warning, or Detail. The following example outputs the date and time of a page request (with a literal space between) to Lasso’s internal error logs with an error level of Detail.

```
[Log_Detail: (Server_Date) + ' ' + (Server_Time)]
```

Table 3: Lasso Error Log Tags

Tag	Description
[Log_Critical]	Logs to Lasso's internal error logs with an error level assignment of Critical. Requires the text to be logged as a parameter. Logging options for this error level are set in Lasso Administration.
[Log_Warning]	Logs to Lasso's internal error logs with an error level assignment of Warning. Requires the text to be logged as a parameter. Logging options for this error level are set in Lasso Administration.
[Log_Detail]	Logs to Lasso's internal error logs with an error level assignment of Detail. Requires the text to be logged as a parameter. Logging options for this error level are set in Lasso Administration.

To log format file errors to the Lasso Service console and Lasso Administration:

Use the [Log_Critical], [Log_Warning], or [Log_Detail] tags. This will log any information contained in the tags in Lasso’s internal error logs with a Lasso error level of Critical, Warning, or Detail. The following example will log a warning to Lasso’s internal logs if an Out Of Memory error occurs while processing the format file.

```
[If: (Error_CurrentError) == (Error_OutOfMemory)]  
  [Log_Warning: 'A memory error occurred while processing this page.']  
[/If]
```

➔ Warning: A memory error occurred while processing this page.

If the Lasso Errors Database and Lasso Service Console options were selected for Warning in the *Monitor > Errors > Setup* page in Lasso Administration,

then this message will be logged and displayed in both the Lasso Service console window and the *Monitor > Errors > Lasso Errors* page in Lasso Administration.

Log Preferences

The tag for setting log preferences is described in *Table 4: Log Preference Tag*. Log preferences can be viewed or changed in the *Monitor > Errors > Setup* section of Lasso Administration.

Table 4: Log Preference Tag

Tag	Description
[Log_SetDestination]	The first parameter specifies a log message level. Subsequent parameters specify the destination to which that level of messages should be logged.

Note: The [Log_SetDestination] tag can only be used by the global administrator. Use an [Auth_Admin] tag to authorize use of this tag.

The first parameter of [Log_SetDestination] requires a log message level. The three available log message levels are detailed in *Table 5: Log Message Levels*.

Table 5: Log Message Levels

Level	Description
Log_Level_Critical	Critical error messages that affect the proper functioning of Lasso Service or requires action by the administrator.
Log_Level_Warning	Informative messages about what actions are being performed by Lasso Service. Generally do not require action by the administrator.
Log_Level_Detail	Detailed messages about the inner workings of Lasso Service. For example, SQL commands are all logged as detail messages.

Subsequent parameters of [Log_SetDestination] require a destination code. The three available destinations available are detailed in *Table 6: Log Destination Codes*.

Table 6: Log Destination Codes

Code	Description
Log_Destination_Console	Messages are logged to the Lasso Service console. Visible on Windows 2000 when Lasso Service is launched as an application and on Mac OS X when the consoleLassoService.command script is used.
Log_Destination_File	Messages are logged to the LassoErrors.txt file which is created in the same folder as Lasso Service.
Log_Destination_Database	Messages are logged to the errors table of the site database which can be viewed in the Monitor > Errors section of Lasso Administration.

To change the log preferences:

Use the [Log_SetDestination] tag to change the destination of a given log message level. In the following example, detail messages are sent to the console and to the errors table of the site database.

```
[Auth_Admin]
[Log_SetDestination: Log_Level_Detail,
  Log_Destination_Database, Log_Destination_Console]
```

To reset the log preferences:

The following three commands reset the log preferences to their default values. Critical errors are sent to all three destinations. Warnings and detail messages are sent only to the console.

```
[Auth_Admin]
[Log_SetDestination: Log_Level_Critical,
  Log_Destination_Console, Log_Destination_Database, Log_Destination_File]
[Log_SetDestination: Log_Level_Warning,
  Log_Destination_Console]
[Log_SetDestination: Log_Level_Detail,
  Log_Destination_Console]
```

File Tags

The [File_...] tags can be used to read and write files on the same machine as Lasso Service. Any text or binary file with an approved file suffix can be manipulated. **Table 7: File Tags** lists the LDML substitution tags that are available to list, inspect, read, write, modify, and delete files. Examples of using the tags are included both in this section and in the *File Upload* section that follows.

Note: See also the section on *File Streaming* for information on how to manipulate files using an object-oriented methodology.

Specifying Paths

There are three different types of paths which can be used with the file tags depending on where the files that are to be manipulated are located.

- **Relative Paths** – Relative paths are specified from the location of the current format file. Relative paths follow the same basic rules as for paths specified within the [Include] tags or within HTML anchor <a> tags except that the ../ construct cannot be used to refer to files in a folder outside the current folder.

For example, the following tag returns the creation date of a file named library.lasso located in the same folder as the current format file.

```
[File_CreationDate: 'library.lasso']
```

→ 11/6/2001 14:30:00

The following tag returns the creation date of a file named library.lasso located in a sub-folder named Includes located in the same folder as the current format file.

```
[File_CreationDate: 'Includes/library.lasso']
```

→ 8/5/2001 15:35:30

Note: The use of relative paths requires that Lasso Service and the Lasso Web server connector be running on the same machine. The file tags only work with files that are located on the same machine as Lasso Service.

- **Absolute Paths** – Absolute paths are specified from the root of the current Web serving folder. Absolute paths always start with a forward slash /. The root of the current Web serving folder is defined by the preferences of the Web server and usually corresponds to the location of the default page that is served when a simple URL such as <http://www.example.com/> is visited.

Relative paths follow the same basic rules as for paths specified within the [Include] tags or within HTML anchor <a> tags except that the ../ construct cannot be used to refer to files in a folder outside the current Web serving folder.

For example, the following tag returns the creation date of a file named index.html located in the root of the Web serving folder.

```
[File_CreationDate: '/index.html']
```

→ 11/3/2001 16:06:15

The following tag returns the creation date of a file named `header.lasso` located in a sub-folder named `Includes` located in the root of the Web serving folder.

```
[File_CreationDate: '/Includes/header.lasso']
```

→ 6/7/2001 8:35:45

Note: The use of relative paths requires that Lasso Service and the Lasso Web server connector be running on the same machine. The file tags only work with files that are located on the same machine as Lasso Service.

- **Mac OS X Fully Qualified Paths** – Fully qualified paths are specified from the root of the file system. They can be used to specify any files on the Web server including those outside of the Web serving root.

In Mac OS X, fully qualified paths are always preceded by three forward slashes `///`. This identifier is used to distinguish fully qualified paths from absolute paths. The root folder `///` corresponds to the root of the file system as defined in the Terminal application (e.g. `cd /`).

For example, the following tag returns the creation date of Lasso Service in Mac OS X.

```
[File_CreationDate: '///Applications/Lasso Professional 7/LassoService']
```

→ 11/3/2001 16:06:15

The following tag returns the creation date of `Admin.LassoApp` located in the default Web serving folder in Mac OS X.

```
[File_CreationDate: '///Library/WebServer/Documents/Lasso/Admin.LassoApp']
```

→ 6/7/2001 8:35:45

Partitions and mounted servers are located in the `///Volumes/` folder. The default Web serving folder for Apache is `///Library/WebServer/Documents/` and for WebSTAR V is `///Applications/4DWebSTAR/WebServer/DefaultSite/`.

- **Windows Fully Qualified Paths** – Fully qualified paths are specified from the root of the file system. They can be used to specify any files on the Web server including those outside of the Web serving root.

In Windows, fully qualified paths are always preceded by the letter name of a partition, a colon, and two forward slashes `C://` or `E://`. Any mounted partition can be referenced in this fashion.

For example, the following tag returns the creation date of Lasso Service from the C: drive in Windows.

```
[File_CreationDate: 'C://Blue World Communications/Lasso Professional 7/LassoService.exe']
```

→ 11/3/2001 16:06:15

The following tag returns the creation date of Admin.LassoApp located in the default Web serving folder from the C: drive in Windows.

[File_CreationDate: 'C://inetPub/WWWRoot/Lasso/Admin.LassoApp']

→ 6/7/2001 8:35:45

Note: The file tags only work with files that are located on the same machine as Lasso Service or are accessible through a mounted file server.

File Suffixes

Any file which is manipulated by Lasso using the file tags must have an authorized file suffix within Lasso Administration. See *Chapter 6: Setting Global Preferences* of the Lasso Professional 7 Setup Guide for more information about how to authorize file suffixes.

By default the following suffixes are authorized within Lasso Administration. Files named with any of these file suffixes can be used with the file tags.

.htm	.html
.inc	.Lasso
.LassoApp	.text
.txt	.uld
.pdf	.xml
.gif	.jpg
.psd	.png
.bmp	.tif
.rbg	.cmyk

Note: If permission has been granted to Access Files Outside of Root for the current user then the file suffix preferences are ignored and files with any file suffix can be manipulated.

Security

The use of file tags is restricted based upon what permissions have been granted in Lasso Administration. Any file operation must pass the following four security checks in order to be allowed.

- **File Tags Enabled** – The desired file tag must be enabled within the *Setup > Settings > Tags* section of Lasso Administration. Tags which are disabled in this section are not available for use by any user other than the global administrator.
- **File Tag Permissions** – The current user must have permission to execute the desired file tag. Permission is granted in the *Setup > Security > Tags* section of Lasso Administration. Permission must be

granted for one of the groups in which the current user belongs or for the AnyUser group.

- **File Permissions** – The current user must have permission to execute the desired file action. Permission is granted in the *Setup > Security > Files* section of Lasso Administration. Permission must be granted for one of the groups to which the current user belongs or for the AnyUser group.
- **Allow Path** – The Allow Path for the current user must allow the file to be accessed. The Allow Path is specified in the *Setup > Security > Files* section of Lasso Administration. Any files in sub-folders of the allowed path can be manipulated using the file tags.
- **File Suffixes** – Discussed above. The file to be operated upon must be named with an approved file suffix.

The Access Files Outside of Root permission specified in the *Setup > Security > Files* section of Lasso Administration allows a user to access file without respect to the allowed path or file suffixes. Any files on the machine hosting Lasso Service can be manipulated.

Mac OS X Note: See the Mac OS X Tips document in the Documentation folder for information about how to configure Mac OS X file permissions.

The global administrator has permission to perform any Lasso actions and is able to access any files on the machine hosting Lasso service without regard to these security settings.

Table 7: File Tags

Tag	Description
[File_Copy]	Copies a file or directory from one location to another. Accepts two parameters, the location of the file or directory to be copied and the new location. Optional -FileOverWrite keyword specifies that the destination file should be overwritten if it exists.
[File_Create]	Creates a new, empty file or a new directory. Accepts one parameter, the location of the file or directory to be created. If the file name ends in a / then a directory is created. Optional -FileOverWrite keyword specifies that the destination file should be overwritten if it exists.
[File_CreationDate]	Returns the creation date of a file. Accepts one parameter, the name of the file or directory to be inspected.
[File_CurrentError]	Reports the last error reported by a file tag. Accepts an optional keyword -ErrorCode that returns the error code rather than the error message.

[File_Delete]	Deletes a file or directory. Accepts one parameter, the name of the file or directory to be deleted.
[File_Exists]	Returns True if the file or directory exists. Accepts one parameter, the name of the file or directory to be inspected.
[File_GetSize]	Returns the size in bytes of a file. Accepts one parameter, the name of the file to be inspected.
[File_IsDirectory]	Returns True if the specified path is a directory. Accepts one parameter, the name of the file or directory to be inspected.
[File_GetLineCount]	Returns the number of lines in a file. Accepts one parameter, the name of the file to be inspected. Optional -FileEndOfLine keyword/value parameter specifies what character represents the end of a line.
[File_ListDirectory]	Returns an array of strings. Each item in the array is the name of one file in the directory. Accepts one parameter, the name of the directory to be listed.
[File_ModDate]	Returns the modification date of a file. Accepts one parameter, the name of the file or directory to be inspected.
[File_Move]	Moves a file or directory from one location to another. Accepts two parameters, the location of the file or directory to be moved and the new location. Optional -FileOverWrite keyword specifies that the destination file should be overwritten if it exists.
[File_Read]	Reads the contents of a file. Accepts one parameter, the name of the file to be read. Two optional parameters -FileStartPos and -FileEndPos define the range of characters which should be read from the file.
[File_ReadLine]	Reads a single line from a file. Accepts two parameters, the name of the file to be read and -FileLineNumber specifying which line of the file to read. An optional keyword/value parameter -FileEndOfLine specifies what character represents the end of lines within the file.
[File_Rename]	Renames a file or directory. Accepts two parameters, the location of the file or directory to be copied and the new name. Optional -FileOverWrite keyword specifies that the destination file should be overwritten if it exists.
[File_SetSize]	Sets the size of the specified file. Accepts two parameters, the name of the file to be modified and the size in bytes which the file should be set to. Any data beyond that size in bytes will be truncated.

[File_Write]	Writes data to the specified file. Accepts two parameters, the name of the file to be written and the data which should be written into the file. Optional -FileOverWrite keyword specifies that the destination file should be overwritten if it exists, otherwise the data specified is appended to the end of the file.
[File_Chmod]	Allows the Unix file permissions of a file to be modified. Requires the name and path to the file to be modified, and a decimal, hex, or octal chmod permission string as parameters. For more information on decimal, hex, and octal chmod permission strings, please see a Unix file system reference. This tag is currently supported on Mac OS X.

See *Appendix A: Error Codes* under the table *File Codes* for a list of error codes and messages which will be returned by the [File_CurrentError] tag.

To list a directory:

- Use the [File_ListDirectory] tag with the path to the directory. An array is returned which can be output using [Loop] ... [/Loop] tags. In the following example the contents of the Web serving folder on a Windows machine is listed by storing the array of files in an array File_Listing and then looping through the array. Each machine will have a different listing depending on what files have been installed in this directory.

```
[Variable: 'File_Listing' = (File_ListDirectory: 'C://InetPub/WWWRoot/')]
[Loop: ($File_Listing->Size)]
  <br>[Output: $File_Listing->(Get: (Loop_Count))]
[/Loop]
```

```
→ <br>default.htm
   <br>default.lasso
   <br>error.lasso
   <br>Images/
   <br>Lasso/
```

Note: The Web serving root on either platform can be listed using [File_ListDirectory: '/'] as long as both Lasso Service and the Web server are hosted on the same machine.

- The number of files in a directory can be counted by simply outputting the size of the array which is returned from [File_ListDirectory]. In the following example, the number of files in the Web serving folder listed above is returned.

```
[Variable: 'File_Listing' = (File_ListDirectory: 'C://InetPub/WWWRoot/')]
[Output: $File_Listing->Size]
```

```
→ 5
```

- More information about each of the files can be returned using the other file tags. The following example shows how to return the size, creation and modification dates of each of the files as well as whether each file is actually a file or a directory. The two directories do not have sizes or date information.

```
[Variable: 'File_Root' = 'C://InetPub/WWWRoot/']
[Variable: 'File_Listing' = (File_ListDirectory: $File_Root)]
[Loop: ($File_Listing->Size)]
  [Variable: 'File_Temp' = $File_Root + $File_Listing->(Get: (Loop_Count))]
  <br>[Output: $File_Temp] [File_GetSize: $File_Temp]
  [File_CreationDate: $File_Temp] [File_ModDate: $File_Temp]
  [File_Exists: $File_Temp] [File_IsDirectory: $File_Temp]
[/Loop]
```

```
➔ <br>default.htm 4325 11/15/2000 14:00:12 11/13/2000 17:26:18 True False
<br>default.lasso 12130 4/11/2000 12:33:29 3/17/2001 11:09:43 True False
<br>error.lasso 393 11/13/2000 11:46:15 11/13/2000 11:50:47 True False
<br>Images/ True True
<br>Lasso/ True True
```

To create a new directory:

A new directory can be created using the [File_Create] tag. The tag creates a directory if the file name specified ends in a slash / character.

- The following tag would create a new directory named files at the root of the Web serving folder.

```
[File_Create: 'files/']
```

- The following tag would create a new directory named files at the root of the Web serving folder using a fully qualified path on Windows 2000.

```
[File_Create: 'C://InetPub/wwwroot/files/']
```

- The following tag would create a new directory named files at the root of the default Apache Web serving folder using a fully qualified path on Mac OS X.

```
[File_Create: '///Library/WebServer/Documents/files/']
```

To create a new file:

- A new file can be created using the [File_Create] tag. The data for the file in the following example comes from a variable File_Contents. The entire file newfile.lasso is written in one step using [File_Write]. If a file of the same name already exists in the specified directory it will be overwritten.

```
[File_Create: 'files/newfile.lasso', -FileOverWrite]
[File_Write: 'files/newfile.lasso', $File_Contents, -FileOverWrite]
```

- The following example shows how to do a safe file write. The code first checks to see if the desired output file is going to overwrite an existing file. A new file is created and the current error is checked using [File_CurrentError]. If no error occurred then the file is written using the [File_Write] tag.

```
[Variable: 'File_Path' = '/files/newfile.lasso']
[If: (File_Exists: $File_Path) == False]
  [File_Create: $File_Path]
  [If: (File_CurrentError) == (Error_NoError)]
    [File_Write: $File_Path, $File_Contents]
  [Else]
    <br>Error - Error Creating File
[Else]
  <br>Error - File already exists
[/If]
```

To import data from a file:

Data can be imported from a file using the [File_ReadLine] tag to read in each line of the file in turn. The lines of the file can then be parsed and stored in a database or shown to a user.

In the following example, each line of the file is assumed to be tab-delimited output from a database which is split into an array and could be later stored into a database. Each line of the file is split into an array Array_Temp and then the array is stored in the array File_Array.

```
[Variable: 'File_Path' = '///Library/WebServer/Documents/import.lasso']
[Variable: 'File_Array' = (Array)]

[If: (File_Exists: $File_Path)]
  [Loop: (File_GetLineCount: $File_Path)]
    [Variable: 'File_Temp' = (File_ReadLine: $File_Path,
      -FileLineNumber=(Loop_Count))]
    [Variable: 'Array_Temp' = $File_Temp->(Split: 't')]
    [$File_Array->(Insert: ($Array_Temp))]
  [/Loop]
[/If]
```

The end result of importing the file is an array File_Array which contains an element for each line of the file. Each element is itself an array that contains an element for each tab-delimited item of data in the specified line.

To report errors while working with files:

Errors can be reported using the [File_CurrentError] tag. This tag works in much the same way as the [Error_CurrentError] tag. The following code creates a file and writes data into it, reporting errors at each step of the process.

```
[File_Create: 'e://files/newfile.lasso', -FileOverWrite]
<br>Error was [File_CurrentError: -ErrorCode]: [File_CurrentError].
[File_Write: 'e://files/newfile.lasso', $File_Contents, -FileOverWrite]
<br>Error was [File_CurrentError: -ErrorCode]: [File_CurrentError].

→ <br>Error was 0: No Error.
   <br>Error was 0: No Error.
```

See *Appendix A: Error Codes* under the table *File Codes* for a list of error codes and messages which will be returned by the [File_CurrentError] tag.

To change the Unix file permissions of a file:

Use the [File_Chmod] tag. The following example changes the Unix file system permissions of a file to -rwxrwxr-x (read, write, and execute permissions for the file owner, read, write, and execute permissions for the file owner group, and read and execute permissions for all other system users) using an octal string.

```
[File_Chmod: 'file.txt', 0777]
```

Line Endings

Files on Mac OS X, Windows, and Linux each have a different standard for line endings. *Table 8: Line Endings* summarizes the different standards.

Table 8: Line Endings

Tag	Description
Mac OS X	Line feed: \n. Each line is ended with a single line feed character.
Windows	Line feed and carriage return: \r\n. Each line is ended with both a line feed and a carriage return character.
Linux	Line feed: \n. Each line is ended with a single line feed character.

Line ending differences are handled automatically by Web servers and Web browsers so are generally only a concern when reading and writing files using the [File_...] tags. The following tips make working with files from different platforms easier.

- The default line endings used by the [File_LineCount] and [File_ReadLine] tags match the platform default. They are \n in Mac OS and Linux, and \r\n in Windows.
- Specify line endings explicitly in the [File_LineCount] and [File_ReadLine] tags. For example, the following tag could be used to get the line count for a file originally created in Linux.

```
[File_LineCount: 'FileName.txt', -FileEndOfLine="\r"]
```

Or, the following tag could be used to get the line count for a file that was originally created on Windows.

```
[File_LineCount: 'FileName.txt', -FileEndOfLine="\r\n"]
```

- Many FTP clients and Web browsers will automatically translate line endings when uploading or downloading files. Always check the characters which are actually used to end lines in a file, don't assume that they will automatically be set to the standard of either the current platform or the platform from which they originated.
- A text editor such as Bare Bones BBEdit can be used to change the line endings in a file from one standard to another explicitly.

File Uploads

Files can be uploaded to Lasso using standard HTML form inputs. Any uploaded files are processed by Lasso and stored in a temporary location. An array [File_Uploads] is provided that returns information about each of the uploaded files. The Lasso developer must write code to move the files to a safe location in the response page to the form in which they were uploaded. The [File_Copy] tag should be used to move uploaded files to a permanent location. Any files left in the temporary location once the format file has finished executing will be deleted.

File Permissions Note: File access permission for All Files is required for a user to upload files. For more information, see *Chapter 8: Setting Up Security* in the Lasso Professional 7 Setup Guide.

HTML Form for File Upload

HTML forms must specify an enctype of multipart/form-data in order for file upload to work. An <input> tag with a type of file must be specified for each file that can be uploaded using the form. The following form includes a single <input> so one file can be uploaded to Lasso.

```
<form action="response.lasso" method="post" enctype="multipart/form-data">
  Select a file: <input type="file" name="upload" value="">
  <br><input type="submit" value="Upload File">
</form>
```

Once the site visitor selects a file using the file control shown in their browser and selects the Upload File button, the format file response.lasso will be called. Within this file the tag [File_Uploads] returns an array of informa-

tion about each of the files uploaded with the form. In this case the array will only contain one item.

Table 9: File Upload Tags

Tag	Description
[File_Uploads]	Returns an array of maps that contain information about any files that were uploaded with the form that triggered the current format file.

Each element of the array returned by the [File_Uploads] tag is a map with the elements defined in *Table 10: [File_Uploads] Map Elements*. If no files were uploaded then [File_Uploads] returns an empty array. Note that each <input> can only be used to upload one file, but multiple <input> tags can be specified in a single form to upload multiple files.

Table 10: [File_Uploads] Map Elements

Element	Description
Upload.Name	The path to the file in the temporary location where it is stored.
Upload.Size	The size of the file in bytes.
Upload.Type	The type of the file.
Upload.RealName	The path to the file which was uploaded on the visitor's machine. This is the same information as is shown to the user in the feedback in the file control.

To display information about the uploaded files:

- Information about the uploaded files can be displayed to the site visitor by looping through the [File_Uploads] array. The following code loops through the array and returns information about each uploaded file on a separate line. The results are shown for a single uploaded file named Picture.gif.

```
[If: (File_Uploads->Size == 0)]
  No files were uploaded.
[Else]
  [Loop: (File_Uploads->Size)]
    [Variable: 'File_Temp' = (File_Uploads->(Get: (Loop_Count)))]
    <br>[Output: $File_Temp->(Find: 'Upload.Name')]
    [Output: $File_Temp->(Find: 'Upload.Size')]
    [Output: $File_Temp->(Find: 'Upload.Type')]
    [Output: $File_Temp->(Find: 'Upload.RealName')]
  [/Loop]
[/If]
```

→
E://WinNT/Temp/Lasso-tmp4.uld 128 image/gif Picture.gif

To move uploaded files to a permanent location:

All of the files which were uploaded will be deleted when the current format file is finished processing. Each uploaded file must be moved to another location in order to prevent it from being deleted. It is recommended that you use the [File_Copy] tag to move uploaded files.

The following code moves each file that was uploaded to a folder located at the path e://uploads/ named upload1.txt, upload2.txt, etc. From there the files can be further manipulated or moved as needed.

```
[Variable: 'Path' = 'e://uploads/']
[If: (File_Uploads->Size == 0)]
  No files were uploaded.
[Else]
  [Loop: (File_Uploads->Size)]
    [Variable: 'File_Temp' = (File_Uploads->(Get: (Loop_Count)))]
    [File_Copy: $File_Temp->(Find: 'Upload.Name'),
      ($Path + 'upload' + (loop_count) + '.txt')]
  [/Loop]
[/If]
```

The code does not return any output if there were no files uploaded.

File Streaming Tags

File streaming tags allow file to be cast as LDML objects and manipulated using member tags. This methodology is more advanced than the [File_...] tags methodology, giving Lasso developers a wide array of file connection modes and types for connecting to files.

Note: All guidelines for specifying file paths, file extensions, line endings, and permissions that were described in the *File Tags* section also apply to the file streaming tags described here.

File Data Type

To use the file streaming methodology, a file must first be cast as an LDML file variable using the [File] tag. This tag is described below.

Table 11: [File] Tag

Tag	Description
[File]	Casts a file as an LDML object, and sets the open and read modes. Requires the name and path to a file as a parameter.

When a file connection is opened using the [File] tag, several different open modes can be used. These modes optimize the file connection for best performance depending on the purpose of the connection. The open modes are described below.

Table 12: File Open Modes

Mode	Description
File_OpenRead	Sets the file connection to read-only.
File_OpenWrite	Sets the file connection to write-only.
File_OpenReadWrite	Sets the file connection to read and write.
File_OpenWriteAppend	Sets the file connection to write and append data.
File_OpenWriteTruncate	Sets the file connection to write and truncate data.

When using the [File] tag, a read mode may also be specified to determine how the file will be read. The read modes are described below.

Table 13: File Read Modes

Read Mode	Description
File_ModeChar	Reads a file character by character.
File_ModeLine	Reads a character line by line.

To cast a file as an LDML object:

Use the [File] tag. The example below casts a local file named myfile.txt as an LDML object in read-only/character mode. Note that no single quotes are used around the open and read mode designators, as they are type constants and not strings.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeChar)]
```

Manipulating File Objects

Once a file has been cast as an LDML file data type, various [File] member tags can be used to manipulate it. These tags can handle file specification, opening, closing, deleting, reading, writing and meta-data for files.

Table 14: File Streaming Tags

Tag	Description
[File->Open]	Opens a new connection to a file. Requires the name and path to a file as a parameter. Optional parameters may include an open mode and read mode, as in the [File] tag. The open mode and read mode set in the initial [File] tag call are used by default if not respecified.
[File->SetMode]	Sets the file read mode for the connection. This can be File_ModeLine for reading a file line by line, or File_ModeChar for reading a file character by character. Defaults to File_ModeChar if not specified.
[File->Read]	Reads data from a file. Requires the integer number of bytes (characters) to read as a parameter. Outputs the file data as bytes.
[File->Write]	Writes string data to a file. Requires the text string to write as a parameter. Two optional comma-delimited integer parameters may also be specified. The first specifies the number of characters of the text string to write, and the second specifies the number of characters in the text string to skip.
[File->SetPosition]	Sets the position of the file's read/write marker. Requires an integer line or character position (depending on mode) as a parameter. All subsequent reads and writes will occur at the given position.
[File->Position]	Returns the current file position. Defaults to 0 if no previous file operations have been performed.
[File->Get]	Returns the current character or line (depending on the file's read mode) at the current file position.
[File->SetSize]	Sets the size of the file. Requires an integer parameter that specifies the size of the file in bytes.
[File->MoveTo]	Moves the file to the new path. Requires a path on the local server as a parameter.
[File->Delete]	Deletes the file and reinitializes the type instance.
[File->Size]	Returns size of the file in bytes.
[File->Name]	Returns the file's name.
[File->Path]	Returns the full internal path to file.
[File->Close]	Closes a connection to a file. This tag should be called whenever a file streaming operation is finished.
[File->IsOpen]	Returns a value of True if the file connection has not been closed.

To read characters from a file:

Use the [File->Read] tag. The file object should be cast with an open mode that permits reading, and with the read mode set to File_ModeChar. The example below reads the first 256 characters of myfile.txt.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeChar)]
[$File->(Read: 256)]
[$File->Close]
```

To read characters from a file starting at a specified position:

Characters can be read starting at a set position using the [File->SetPosition] tag before the [File->Read] tag. The example below reads 240 characters starting at character number 16.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeChar)]
[$File->(SetPosition: 16)]
[$File->(Read: 240)]
[$File->Close]
```

To read lines from a file:

Use the [File->Read] tag. The file object should be cast with an open mode that permits reading, and with the read mode set to File_ModeLine. The example below reads the first 4 lines of myfile.txt.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeLine)]
[$File->(Read: 4)]
[$File->Close]
```

To read lines from a file starting at a specified position:

Lines can be read starting at a set position using the [File->SetPosition] tag before the [File->Read] tag. The example below reads 6 lines starting at line number four.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeLine)]
[$File->(SetPosition: 4)]
[$File->(Read: 6)]
[$File->Close]
```

To reset the read mode during file operations:

Use the [File->SetMode] tag to change the read mode. The example below starts in File_ModeLine mode, reads the first line of myfile.txt, moves to line five, changes to File_ModeChar mode, and then reads the next 16 characters.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeLine)]
[$File->(Read: 1)]
[$File->(SetPosition: 5)]
[$File->(SetMode: File_ModeChar)]
```

```
[File->(Read: 16)]
[File->Close]
```

To write text to a file:

Use the [File->Write] tag. The file object should be cast with an open mode that permits writing. The example below adds the text `This is some text` after the fifth line of the file.

```
[Var:'File'=(File: 'myfile.txt', File_OpenWrite, File_ModeLine)]
[File->(SetPosition: 5)]
[File->(Write:'this is some text')]
[File->Close]
```

To write part of a string variable to a file:

Use the [File->Write] tag with the optional size and offset integer parameters. This is useful for truncating part of an existing string variable on-the-fly before writing it to the file. The example below adds the text `five` to the file out of a pre-defined string variable with a value of `There are five elements`.

```
[Var:'Text'='There are five elements']
[Var:'File'=(File: 'myfile.txt', File_OpenWrite, File_ModeLine)]
[File->(Write: $Text, 4, 10)]
[File->Close]
```

To return information about a file:

The [File->Name], [File->Path], and [File->Size] tags can be used to output the name, path, and size (in kilobytes) of a file. The example below outputs the file name, path, and size delimited by HTML line breaks.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeChar)]
[File->Path]<br>
[File->Name]<br>
[File->Size]<br>
[File->Close]
```

To move a file to a different folder:

Use the [File->MoveTo] tag. The following examples moves the local `myfile.txt` file to a different folder on a Mac OS X hard drive.

```
[Var:'File'=(File: 'myfile.txt', File_OpenRead, File_ModeChar)]
[File->(MoveTo:'///Library/WebServer/Documents/myfile.txt')]
[File->Close]
```

21

Chapter 21

Error Control

This chapter documents the methods Lasso uses to report errors and the tags available in LDML to capture and respond to errors.

- *Overview* provides definitions of the types of errors Lasso reports and the methods which can be used to capture and respond to them.
- *Error Messages* documents the built-in error messages in Lasso.
- *Custom Error Page* explains how to override the built-in error messages with a custom error page.
- *Error Pages* documents how to create action specific error pages.
- *Error Tags* documents the [Error_...] process and substitution tags that can be used to report custom or standard errors and for basic error handling within a format file.
- *Error Handling* documents the [Protect], [Fail], and [Handle] tags for advanced error handling within a format file.

Overview

Responding to errors gracefully is the hallmark of good programming. Errors in Lasso run the gamut from expected errors such as a database search that returns no records to syntax errors that require fixing before a page will even process. Lasso provides tools to manage errors at several different levels which can act redundantly to ensure that no errors will be missed.

The following lists the types of errors that can occur in or are reported by Lasso. This chapter includes instructions for how to handle each of these types of errors.

Error Types

- **Web Server Errors** include file not found errors and access violations in realms. These will be reported with standard HTTP response codes, e.g. 404 for File Not Found.
- **Syntax Errors** include misspellings of tag names, missing delimiters, and mismatched data types. Lasso will return an error message rather than the processed format file if it encounters a syntax error.
- **Action Errors** include misspellings of database names, table names, or fields names and other problems specifying database actions. The database action cannot be performed until the errors are corrected.
- **Action Results** can be reported as errors by Lasso. For example if no records were found after performing a search.
- **Database Errors** are generated by the data source application and include data type mismatches, missing required field values, and others. Lasso will report the error which was returned from the data source application without modification.
- **Logical Errors** are problems that cause a page to process unexpectedly even though the syntax of the code is correct. These include infinite loops, missing cases, and assumptions about the size or composition of a found set.
- **Security Violations** are not strictly errors, but are attempts to perform database actions or file accesses which are not allowed by the permissions set for the current user. These include permissions to perform database actions, privileges to add users to groups, permissions to use specific tags, and specific permissions to use the file tags.
- **Installation Problems** can also result in error messages if a Lasso Web server connector is improperly configured or Lasso Service is unavailable.
- **Operating System Errors** can also be reported by Lasso if they occur. Lasso will report the error without modification.

Some errors are more serious than others. Pages will not be processed at all if they contain syntax errors or if there are installation problems which prevent Lasso Service from being accessed. Other errors are commonly encountered in the normal use of a Web site. Most database errors and security violations are handled by simple means such as showing a No Records Found message or displaying a security dialog box to prompt the user for a username and password.

There are five mechanisms for handling errors which are detailed in this chapter. These mechanisms can be used singly or in concert to provide comprehensive error handling.

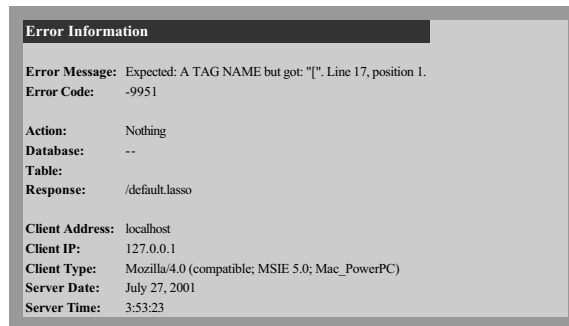
Error Control Types

- Automatic **Error Messages** are generated by Lasso in response to unhandled errors. These error messages are sufficient for many developers. However, much of this chapter is devoted to instructions for how to avoid showing these automatic error messages to Web site visitors.
- A **Custom Error Page** allows the automatic error page to be replaced by a custom page. The built-in error message will be reported at the end of the site-specific custom error page.
- **Action Specific Error Pages** allow error pages to be specified based on specific database actions. Multiple error pages can be specified so that different pages are returned based on what error occurs.
- **Error Tags** allow action and logical errors and security violations to be handled within a format file.
- **Error Handling** tags allow advanced error handling to be built into format files. These techniques allow error handling routines to be built into a page without disrupting the normal processing of a page if no errors occur.

Error Messages

By default, Lasso displays the error message displayed in *Figure 1: Built-In Error Message* when an error occurs processing a page. Lasso will display this page as a result of any of the errors in the *Error Types* list above. If a syntax error is reported, the built-in error message provides a pointer to what code the syntax error is contained in. If a database or action error is reported, the built-in error message provides information about what database action was performed when the error occurred.

Figure 1: Built-In Error Message



The standard built-in error message will be replaced by a custom error page if one is defined. See the next section *Custom Error Page* for more details.

The simple error message in *Figure 2: Lasso Service Error Message* is displayed when Lasso Service cannot be contacted by a Lasso Web server connector. No processing can happen without Lasso Service. This message will be displayed if Lasso Service is quit or restarted while the Web server application is still running.

Figure 2: Lasso Service Error Message

Lasso Error

Lasso Connector could not communicate with Lasso Service.

Security violations result in an appropriate HTTP response being sent to the Web client to ask the site visitor for authentication information. An authentication dialog like that shown in *Figure 3: Authentication Dialog* is presented to the visitor. If they enter a valid username and password then processing proceeds as normal. If they enter an invalid username and password then the standard built-in error message will be shown with details about the security violation.

Figure 3: Authentication Dialog



Custom Error Page

A custom error page can be defined which will be displayed to the site visitor rather than the built-in error message described in the previous section.

Figure 4: Custom Error Page

```

Error: No error
Code: 0

Response: /error.lasso
Path: /
Local: ///Library/WebServer/Documents/error.lasso
Realm:
Referrer:

Date: Tuesday, February 5, 2002
Time: 11:30:37 AM
Version: Mac OS X 5.0.0b15

array: (pair: (-nothing)=()), (pair: (-operatorlogical)=(and)), (pair: (-maxrecords)=(50)), (pair: (-skiprecords)=
(0))

Accept: */*
Accept-Language: en
Authorization: Basic YWRtaW5pc3RyYXRvcj0u23JlZW4u
Connection: Keep-Alive
Extension: Security/Remote-Password
Host: localhost
UA-CPU: PPC
UA-OS: MacOS
User-Agent: Mozilla/4.0 (compatible; MSIE 5.12; Mac_PowerPC)

Apple Stock: 25.75

```

The custom error page must be named `error.lasso`. Each Web site or virtual host which has a distinct Web serving folder can have a custom error page. The `error.lasso` file should be placed in the root of the Web serving folder for a specified Web site or virtual host. Usually, this is the same folder as the default page which is served to site visitors who don't request a specific page from the Web site.

To define a custom error page:

- 1 Create a file named `error.lasso` which includes the default error message to be displayed to site visitors.
- 2 All image links and URLs within the custom error page should be specified as absolute paths from the root of the Web serving folder. The following `` tag contains a reference to `picture.gif` contained in the `images` folder.

```

```

- 3 Place the `error.lasso` file in the root of the Web serving folder for the Web site which is being customized. The file should be accessible by loading the following URL.

```
http://www.example.com/error.lasso
```

Note: The built-in error page will not be displayed if a custom error page is defined. `[Error_CurrentError]` and `[Error_CurrentError: -Errorcode]` should appear on all custom error pages. If desired, other tags can be used to provide information about what database action was being attempted.

To test a custom error page:

A properly placed `error.lasso` file can be tested by loading it with each of the following URLs.

- The first URL loads the page directly. This confirms that the `error.lasso` file is located in the right folder.
`http://www.example.com/error.lasso`
- The second URL will cause an error in Lasso that should return the custom error page. A page not found error will be returned since the file `fakepage.lasso` is not present on the Web server.
`http://www.example.com/Action.Lasso?-response=fakepage.lasso`

Error Pages

A custom error page can be specified in any HTML form or URL based Lasso action using the `-ResponseAnyError` command tag. The `-ResponseRequiredFieldMissingError` tag can be used to trap for missing values which are flagged with the `-Required` command tag. The `-ResponseSecurityError` can be used to trap for security permissions violations.

If an error occurs and no `-Response...` tag is specified then the default error message or a custom error page is returned as documented in the previous section *Custom Error Page*. The details of the Lasso action can be retrieved in the error page and the specific error message which triggered the error page can be returned using `[Error_CurrentError]`.

Neither of the response command tags function within `[Inline] ... [/Inline]` based Lasso actions. Instead, errors should be handled directly within the `[Inline] ... [/Inline]` tags using the techniques outlined in the *Error Tags* and *Error Handling* sections that follow.

Table 1: Error Response Tags

Tag	Description
<code>-ResponseAnyError</code>	Specifies the page to return if any error occurs and no specific error page for that error is specified.
<code>-ResponseReqFieldMissingError</code>	Specifies the page to return if a name/value pair preceded by a <code>-Required</code> command tag does not have a value. Synonyms include <code>-ResponseRequiredFieldMissingError</code> , <code>-ResponseReqColumnMissingError</code> , and <code>-ResponseRequiredColumnMissingError</code> .
<code>-ResponseSecurityError</code>	Specifies the page to return if the current user does not have permission to perform the requested action.

Error Tags

The `[Error_...]` tags in LDML allow custom errors to be reported and provide access to the most recent error that was reported by the code executing in the current format file. This allows the developer to check for specific errors and respond if necessary with an error message or code to correct the error.

Lasso maintains a single error code and error message that is set by any tag which reports an error. The error code and error message should be checked immediately after a tag that may report an error. If any intervening tags report errors then the error code and error message will be lost.

Custom errors can be created using the `[Error_SetErrorMessage]` and `[Error_SetErrorCode]` tags. Once set, the `[Error_CurrentError]` tag or `[Error_Code]` and `[Error_Msg]` tags will return the custom error code and message. A developer can utilize these tags to incorporate both built-in and custom error codes into the error recovery mechanisms for a site.

Table 2: Error Tags

Tag	Description
<code>[Error_CurrentError]</code>	Returns the current error message. Optional <code>-ErrorCode</code> parameter returns the current error code.
<code>[Error_Code]</code>	Returns the current error code.
<code>[Error_Msg]</code>	Returns the current error message.
<code>[Error_SetErrorCode]</code>	Sets the current error code to a custom value.
<code>[Error_SetErrorMessage]</code>	Sets the current error message to a custom value.

To display the current error in a format file:

- Use the `[Error_Msg]` tag and the `[Error_Code]` tag. The following code will display a short error message.

The current error is `[Error_Code]`: `[Error_Msg]`.

If the code on the page is executing normally and there is no current error to report then the code will return.

→ The current error is 0: No Error.

- Use the `[Error_CurrentError]` tag with the optional `-ErrorCode` keyword. The following code will display a short error message.

The current error is `[Error_CurrentError: -ErrorCode]`: `[Error_CurrentError]`.

If the code on the page is executing normally and there is no current error to report then the code will return.

→ The current error is 0: No Error.

To set the current error in a format file:

The current error code and message can be set using the [Error_SetErrorCode] and [Error_SetErrorMessage] tags. These tags will not affect the execution of the current format file, but will simply set the current error so it will be returned by the [Error_CurrentError] tag.

In the following example, the error message is set to
A custom error occurred and the error code is set to -1.

```
[Error_SetErrorMessage: 'A custom error occurred']
[Error_SetErrorCode: -1]
```

The [Error_CurrentError] tag now reports this custom error when it is called later in the page, unless any intervening code changed the error message again.

The current error is [Error_CurrentError: -ErrorCode]: [Error_CurrentError].

→ The current error is -1: A custom error occurred.

The remainder of the [Error_...] tags provide shortcuts for reporting standard errors or checking what error is being reported by Lasso so appropriate steps can be taken. The [Error_...] tags available in LDML are described in *Table 3: Error Type Tag*. An example of how to respond to a particular error message follows.

These tags can be used with the [Error_SetErrorCode] and [Error_SetErrorMessage] tags to generate standard errors. If a page has code which deals with an “Add Error” for example, that code can be triggered by an [Inline] that reports an “Add Error” or by setting the current error to an “Add Error” explicitly using the [Error_SetErrorCode] and [Error_SetErrorMessage] tags as shown in the following code.

```
[Error_SetErrorCode: (Error_AddError: -ErrorCode)]
[Error_SetErrorMessage: (Error_AddError)]
```

Table 3: Error Type Tags

Tag	Description
[Error_AddError]	An error occurred during an -Add action.
[Error_DatabaseConnection Unavailable]	A connection to the specified Lasso data source connector for the current database cannot be established.
[Error_DatabaseTimeout]	The connection to the Lasso data source connector timed out.
[Error_DeleteError]	An error occurred during a -Delete action such as if an invalid -KeyField or -KeyValue was specified.
[Error_FieldRestriction]	An error reported by the Lasso data source connector that a field cannot be modified. Synonym is [Error_ColumnRestriction].
[Error_FileNotFound]	The specified file in an [Include] tag or -Response... tag cannot be found.
[Error_InvalidDatabase]	The specified database is not configured within Lasso Administration.
[Error_InvalidPassword]	The password for the specified username is invalid.
[Error_InvalidUsername]	The specified username cannot be found in the users database within Lasso security.
[Error_NoError]	The code has been executed successfully. This error code represents the lack of an error.
[Error_NoPermission]	The current user does not have permission to perform the requested database action.
[Error_OutOfMemory]	Lasso encountered an internal out of memory error that prevents the current page from processing.
[Error_RequiredFieldMissing]	A value was not specified for an HTML form or URL parameter preceded by a -Required command tag. Also [Error_RequiredColumnMissing].
[Error_UpdateError]	An error occurred during an -Update action such as if an invalid -KeyField or -KeyValue was specified.

Note: In prior versions of Lasso an [Error_NoRecordsFound] tag was defined. This tag has been deprecated in favor of checking whether the [Found_Count] is equal to zero to check if no records were found.

To check for a specific error within [Inline] ... [/Inline] tags:

Use a conditional expression in [If] ... [/If] tags to compare [Error_CurrentError] with the specific error type tag you want to check. In the following example, a different message is displayed if no records were found after a -FindAll action or if the requested database was not found.

```
[Inline: -Database='Contacts', -Table='People', -FindAll]
  [If: (Error_CurrentError) == (Error_InvalidDatabase)]
    The database Contacts is not valid.
  [Else: (Error_CurrentError) == (Error_NoPermission)]
    You don't have permission to search Contacts.
  [Else: (Found_Count) == 0]
    No records were found in Contacts.
  [Else]
    ... Display Found Set Here ...
[/If]
[/Inline]
```

Error Handling

LDML includes powerful error handling tags that allow areas of a page to be protected. Error-specific handlers are called if any errors occur in a protected area of a page. These tags allow comprehensive error handling to be built into a page without disturbing the code of the page with many conditionals and special cases.

Table 4: Error Handling Tags

Tag	Description
[Fail]	Halts execution of the current page or [Protect] ... [/Protect] block. Takes two parameters: an integer error code and a string error message.
[Fail_If]	Conditionally halts execution of the current page or [Protect] ... [/Protect] block. Takes three parameters: a conditional expression, an integer error code, and a string error message.
[Handle] ... [/Handle]	Conditionally executes after the code in the current container tag or format file is completed or a [Fail] tag is called. Takes a conditional expression as a parameter.
[Handle_Error] ... [/Handle_Error]	Functions the same as [Handle] ... [/Handle] except that the contents is executed only if an error was reported in the surrounding [Protect] ... [/Protect] tags.
[Protect] ... [/Protect]	Container tag that protects a portion of a page. If code inside the container throws an error or a [Fail] tag is executed inside the container then the error is not allowed to propagate outside the protected block.

Handle Tags

The [Handle] ... [/Handle] tags are used to surround a block of code that will be executed after the current code segment is completed. The opening [Handle] tag takes a single parameter which is a conditional expression. If the conditional expression returns True then the code in the [Handle] ... [/Handle] tags is executed. Every [Handle] tag is given a chance to execute in the order they were specified so multiple [Handle] ... [/Handle] tags can be executed.

[Handle] ... [/Handle] tags will not be executed if a syntax error occurs while Lasso is parsing a page. When Lasso encounters a syntax error it returns an error page instead of processing the code on a page.

[Handle] ... [/Handle] tags will be executed if a logical error occurs while Lasso is processing a page. However, the result of the page will be an error message rather than the output of the page. Code within the [Handle] ... [/Handle] tags can redirect the user to another page using [Redirect_URL] or can replace the contents of the page being served.

There are two ways to use [Handle] ... [/Handle] tags within a format file:

- When used on their own in a format file, the code inside the [Handle] ... [/Handle] tags will be conditionally executed after all the rest of

the code in the format file has completed. [Handle] ... [/Handle] tags can be used to provide post-processing code for a format file.

- When used within any LDML container tag, the code inside the [Handle] ... [/Handle] tags will be conditionally executed after the closing container tag. [Handle] ... [/Handle] tags will most commonly be used within [Protect] ... [/Protect] tags to provide error handling.

To specify code to execute if a format file reports an error:

Place [Handle] ... [/Handle] tags with a check for [Error_CurrentError] anywhere in a page, but not inside any other container tags. In the following example, the opening [Handle] tag checks if [Error_CurrentError] is not equal to [Error_NoError]. The contents of the page which is being returned to the visitor is replaced by a custom error message if an error has occurred.

```
[Handle: (Error_CurrentError) != (Error_NoError)]
  [Var: '__html_reply__' = '<hr>' +
    'An error occurred while processing this page:' +
    (Error_CurrentError: -ErrorCode) + ': ' + (Error_CurrentError) + '.']
[/Handle]
```

To output debugging messages at the end of a format file:

Place [Handle] ... [/Handle] tags throughout a page that check to see if a variable named Debug equals True. The contents of the [Handle] ... [/Handle] tags will only be executed if it does. Note that the [Handle] ... [/Handle] tags can only contain static messages because they do not execute within the flow of the page.

```
[Var: 'Debug'=True]

[Handle: (Variable: 'Debug') == True]
  <p>Debugging Message
[/Handle]
```

Note: If a syntax or logical error occurs while processing the page then this handle code will execute, but the results may not be visible since the default error page will be returned in place of the processed page contents.

To specify code to post-process a format file:

Place [Handle] ... [/Handle] tags with a condition of True anywhere in the format file, but not within any other container tags. The contents of the [Handle] ... [/Handle] will execute after the rest of the format file has executed.

In the following example, the global variable which contains the text of the page which will be sent to the site visitor __html_reply__ is modified using [String_ReplaceRegExp] so that all occurrences of the words Blue World are wrapped with tags that make them blue.


```

<?LassoScript
// This LassoScript implements a post-processor that makes all occurrences
// of the words Blue World within the current format file blue.
Handle: True; // Unconditionally execute handler.
Variable: '__html_reply__' = (String_ReplaceRegExp: $__html_reply__,
    -Find='([Bb]lue+[Ww]orld)',
    -Replace='<font color="blue">\1</font>');
/Handle;
?>

```

Fail Tags

The `[Fail]` tag allows an error to be triggered from within LDML code. The two parameters of the tag are the integer error code and string error message of the error to be reported. Use of the `[Fail]` tag immediately halts execution of the current page and starts execution of any `[Handle] ... [/Handle]` tags contained within.

The `[Fail]` tag can be used in the following ways:

- To report an unrecoverable error. Just as Lasso automatically halts execution of a format file when a syntax error or internal error is encountered, LDML code can use the `[Fail]` tag to report an error which cannot be recovered from.
`[Fail: -1, 'An unrecoverable error occurred']`
- To trigger immediate execution of the page's `[Handle] ... [/Handle]` tags. If an error is handled by one of the `[Handle] ... [/Handle]` tags specified in the format file (outside of any other container tags) then the code within the `[Handle] ... [/Handle]` tags will be executed.
- To trigger immediate execution of a `[Protect] ... [/Protect]` block's `[Handle] ... [/Handle]` tags. See the next section *Protect Tags* for details.

To report a standard Lasso error:

Use the appropriate `[Error_...]` tag to return the error code and error message for any of Lasso's standard errors. In the following example a No Records Found error is triggered.

```
[Fail: (Error_NoRecordsFound: -ErrorCode), (Error_NoRecordsFound)]
```

To conditionally execute a `[Fail]` tag:

`[Fail_If]` allows conditional execution of a `[Fail]` without using a full `[If] ... [/If]` tag. The first parameter to `[Fail_If]` is a conditional expression. The last two parameters are the same integer error code and string error message as in the `[Fail]` tag. In the following example the `[Fail_If]` tag is only executed if the `[Found_Count]` is 0.

```
[Fail_If: (Found_Count == 0),  
  (Error_NoRecordsFound: -ErrorCode), (Error_NoRecordsFound)]
```

Protect Tags

The `[Protect] ... [/Protect]` tags are used to catch any errors that occur within the code surrounded by the container tags. They create a protected environment from which errors cannot propagate to the page itself. Even if an internal error is reported by Lasso it will be caught by the `[Protect] ... [/Protect]` tags allowing the rest of the page to execute successfully. Any `[Fail]` or `[Fail_If]` tags called within `[Protect] ... [/Protect]` tags will halt execution only if the code is contained within the `[Protect] ... [/Protect]` tags. Any `[Handle] ... [/Handle]` tags contained within the `[Protect] ... [/Protect]` tags will be conditionally executed. The format file will continue executing normally after the closing `[/Protect]` tag.

The `[Protect] ... [/Protect]` tags can be used for the following purposes:

- To protect a portion of a page so that any errors that would normally result in an error message being displayed to the user are instead handled in the internal `[Handle] ... [/Handle]` tags.
- To provide advanced flow control in a page. Code within the `[Protect] ... [/Protect]` tags is executed normally until a `[Fail]` tag is encountered. The code then jumps immediately to the internal `[Handle] ... [/Handle]` tags.

To protect a portion of a page from logical errors:

Wrap the portion of the page that needs to be protected in `[Protect] ... [/Protect]` tags. Any internal errors that Lasso reports will be caught by the `[Protect] ... [/Protect]` tags and not reported to the end user. `[Handle] ... [/Handle]` should be included to handle the error if necessary.

In the following LassoScript an attempt is made to set the global map `[Tags]` to `Null`. This would have the effect of removing all tags from LDML so their operation is not allowed. Instead, Lasso reports a logical error. Since this code is executed within `[Protect] ... [/Protect]` tags no error is reported, but the `[Protect] ... [/Protect]` tags exit silently and the format file resumes executing after the end of the LassoScript.

```
<?LassoScript  
  Protect;  
    $Tags = Null;  
  /Protect;  
?>
```

To use the [Protect] ... [/Protect] tags with custom errors:

The following example shows [Protect] ... [/Protect] tags which surround code that contains several [Fail_If] statements with custom error codes -1 and -2. A pair of [Handle] ... [/Handle] tags inside the [Protect] ... [/Protect] tags are set to intercept either of these custom error codes. These [Handle] ... [/Handle] tags will only execute if one of the [Fail_If] tags executes successfully.

```
[Protect]
...
[Fail_If: ($ConditionOne == True), -1, 'Custom error -1']
...
[Fail_If: ($ConditionTwo == True), -2, 'Custom error -2']
...
[Handle: ((Error_CurrentError: -ErrorCode) == -1)]
... Handle custom error -1 ...
[/Handle]
[Handle: (Error_CurrentError: -ErrorCode) == -2)]
... Handle custom error -2 ...
[/Handle]
[/Protect]
```


22

Chapter 22

Control Tags

This chapter documents tags that allow format files to be scheduled for execution and tags that allow low-level access to Lasso's internal variables.

- *Authentication Tags* details the tags that allow security settings to be modified.
- *Administration Tags* details the tags that provide access to Lasso's environment.
- *Scheduling Events* documents how to use the [Event_Schedule] tag to schedule the loading of format files.
- *Process Tags* documents how to programmatically control what code is processed by Lasso, and how to pause Lasso execution.
- *Configuration Tags* allow the configuration of Lasso to be inspected.
- *Page Variables* documents the internal variables Lasso uses while processing a page and tags that allow access to them.
- *Null Data Type* documents the base data type and the member tags common to all data types in LDML.
- *Format File Execution Time Limit* describes the built-in time limit on the length of time that format files are allowed to execute.

Authentication Tags

The authentication tags can be used to ensure that all of the code in a page is run by a registered user in Lasso Security or the global administrator. The authentication tags work by performing the following tasks when they are executed:

- 1 Check the current username and password stored in the client browser. If the username and password meet the requirements of the authentication tag used, then the page is served normally.
- 2 Otherwise, a browser authentication dialog box is shown to the visitor.
- 3 If the client enters a valid username and password, then the page is served normally.
- 4 Otherwise, the visitor is either prompted for a username and password again or is shown an error. The actual behavior is determined by the Web browser software.

Table 1: Authentication Tags

Tag	Description
[Auth]	Allows only configured Lasso users to view the page. Prompts for a username and password if the current visitor has not provided a valid username and password.
[Auth_Admin]	Allows only the global administrator to view the page. Prompts for a username and password if the current visitor is not the global administrator.
[Auth_Group]	Allows only configured Lasso users in a specified group in Lasso Security to view the page. Requires the name of a Lasso group (or an array of group names) as a parameter. Prompts for a username and password if the current visitor has not provided a valid username and password.
[Auth_User]	Allows a single specified user in Lasso Security to view the page. Requires the name of a Lasso user (or an array of user names) as a parameter. Prompts for a username and password if the current visitor has not provided a valid username and password.
[Auth_Custom]	Allows a single user with a specified username and password to view the page. Requires a custom username (or array of usernames), password (or array of corresponding passwords), and realm as parameters. The custom username does not have to be configured in Lasso Security.

To prompt a visitor for authentication:

Use the [Auth] tag at the top of a format file. Each visitor will need to enter a username and password which is configured within Lasso Administration in order to view the contents of the Web page.

[Auth]

This tag can be used to ensure that only configured users visit a format file. No Anonymous users or users who do not have a valid username and password will be allowed.

To restrict a page so only the global administrator can view it:

Use the [Auth_Admin] tag at the top of a format file. Each visitor will need to enter the global administrator's username and password in order to view the contents of the Web page.

```
[Auth_Admin]
```

This can be used to hide format files which provide status information that only the global administrator should be able to read or to protect custom format files that allow aspects of a Web site to be administered.

To restrict a page to users in a specific Lasso group:

Use the [Auth_Group] tag with the name of the desired group as a parameter. Each visitor will need to enter a username and password that belongs to the specified Lasso group in order to view the contents of the Web page.

```
[Auth_Group: 'Lasso_Group_Name']
```

To restrict a page to a specific Lasso user:

Use the [Auth_User] tag with the name of the desired user as a parameter. Each visitor will need to enter the username and password of the Lasso user specified in order to view the contents of the Web page.

```
[Auth_User: 'Lasso_User_Name']
```

To restrict a page to a custom username and password:

Use the [Auth_Custom] tag with a custom username, password, and realm as parameters (the parameters must be entered in this order). Each visitor will need to enter this username and password in order to view the contents of the Web page.

```
[Auth_Custom: 'Custom_User_Name', 'Custom_Password', 'Custom_Realm']
```

This tag is useful for authenticating a user that is not necessarily configured in Lasso Security. The custom realm will be displayed in the authentication dialog box when the user logs in, and can be used in conjunction with other realms on the Web server.

Administration Tags

Lasso Security is generally configured through the Lasso Administration interface and related LassoApps. However, Lasso also provides a number of tags that allow the security settings to be modified from within format files. These tags are summarized in *Table 2: Administration Tags*. See *Chapter 8: Setting Up Security* of the Lasso Professional 7 Setup Guide for more information about users and groups.

Table 2: Administration Tags

Tag	Description
[Admin_CurrentUsername]	Returns the name of the current user whose permissions are being used to run the page or inline. Returns nothing for the anonymous user.
[Admin_CurrentGroups]	Returns an array of Lasso group names that the current user belongs to.
[Admin_ChangeUser]	Changes a user's password. Requires a valid Lasso username, old password, and new password as parameters. This tag can be called by any configured Lasso user.
[Admin_CreateUser]	Creates a new user with the specified password. Requires a new username and password as parameters. This tag can be called by any group administrator.
[Admin_GroupAssignUser]	Assigns the specified user to the group. Requires the name of a Lasso group and a Lasso user as parameters. This tag can only be called by a group administrator for the group.
[Admin_GroupListUsers]	Lists the users who belong to the group. This tag can only be called by a group administrator for the group.
[Admin_GroupRemoveUser]	Removes the specified user from the group. This tag can only be called by a group administrator for the group.
[Admin_ListGroups]	Lists the groups for which a group administrator has privileges.
[Admin_RefreshSecurity]	Refreshes cached security settings. This tag can be used only by the Lasso global administrator.
[Admin_ReloadDataSource]	Reloads a Lasso Data Source Connector. Requires the internal name or ID number of a Lasso Data Source Connector as a parameter (as shown in Lasso Administration). This tag can be used only by the Lasso global administrator.
[Admin_LassoServicePath]	Returns the file path to Lasso Service. This tag can be used only by the Lasso global administrator.

To return the current username within a format file:

Use the [Admin_CurrentUserName] tag. The following example displays the current Lasso user being used within an [Inline] tag.

```
[Inline: -Username='John_Doe', -Password='MyPassword']
  [Admin_CurrentUserName]
[/Inline]
```

→ John_Doe

To change the password for a user:

Use the [Admin_ChangeUser] tag. The tag takes three parameters: the username and password of an existing user, and the new password for the user. The following example changes the password for John_Doe to MyPassword from MyOldPassword. The tag returns True if the change was successful.

```
[Admin_ChangeUser: 'John_Doe', 'MyOldPassword', 'MyPassword']
```

→ True

To list the groups the current user belongs to:

Use the [Admin_CurrentGroups] tag. The following example lists the groups that John_Doe belongs to.

```
[Inline: -Username='John_Doe', -Password='MyPassword']
  [Admin_CurrentGroups]
[/Inline]
```

→ (Array: (AnyUser), (Johns_Group))

To list the groups the current user can administer:

Use the [Admin_ListGroups] tag. The following example lists the groups that John_Doe can administer. The username and password for John_Doe are specified using -Username and -Password command tags in [Inline] ... [/Inline] tags surrounding the call to [Admin_ListGroups]. Since John_Doe is the administrator of one group, Johns_Group, this one name is returned in a single element array.

```
[Inline: -Username='John_Doe', -Password='MyPassword']
  [Admin_ListGroups]
[/Inline]
```

→ (Array: (Johns_Group))

To list the users that belong to a group:

Use the [Admin_GroupListUsers] tag. The following example lists the users that belong to the group named Johns_Group. The username and password

for an administrator of Johns_Group is specified in the [Inline] ... [/Inline] tags surrounding the call to [Admin_GroupListUsers]. An array of usernames is returned.

```
[Inline: -Username='John_Doe', -Password='MyPassword']
  [Admin_GroupListUsers: 'Johns_Group']
[/Inline]
```

→ (Array: (John_Doe), (Jane_Doe), (Tex_Surname), (Bob_Peoples))

To create a new user:

Use the [Admin_CreateUser] tag. The following example creates a new user Joe_Random with the password 1234. Joe_Random will not have any permissions beyond those assigned to AnyUsers until he is assigned to a group. Since only a group administrator can create new users, the username and password for John_Doe are specified in the [Inline] ... [/Inline] tags surrounding the call to [Admin_CreateUser]. The tag returns True if the user was created or False if the username already exists.

```
[Inline: -Username='John_Doe', -Password='MyPassword']
  [Admin_CreateUser: 'Joe_Random', '1234']
[/Inline]
```

→ True

To add a user to a group:

Use the [Admin_GroupAssignUser] tag. The following example adds the user Joe_Random to the group Johns_Group. This tag can only be called by a group administrator for Johns_Group so the username and password for John_Doe are specified in the [Inline] ... [/Inline] tags surrounding the call to [Admin_GroupAssignUser]. The tag returns True if the user is successfully added to the group.

```
[Inline: -Username='John_Doe', -Password='MyPassword']
  [Admin_GroupAssignUser: 'Johns_Group', 'Joe_Random']
[/Inline]
```

→ True

To remove a user from a group:

Use the [Admin_GroupRemoveUser] tag. The following example removes the user Joe_Random from the group Johns_Group. This tag can only be called by a group administrator for Johns_Group so the username and password for John_Doe are specified in the [Inline] ... [/Inline] tags surrounding the call to [Admin_GroupRemoveUser]. The tag returns True if the user is successfully removed from the group.

```
[Inline: -Username='John_Doe', -Password='MyPassword']
  [Admin_GroupRemoveUser: 'Johns_Group', 'Joe_Random']
[/Inline]
```

→ True

To reload a data source connector:

Use the [Admin_ReloadDataSource] tag. This tag is useful if a data source has been modified and Lasso needs to be refreshed to see the new changes, and requires Lasso global administrator permission to use. The example below refreshes the Lasso MySQL data source connector, which has an internal ID number of 1 according to the *Setup > Data Sources > Connectors* section of Lasso Administration.

```
[Admin_ReloadDataSource: 1]
```

To refresh Lasso Security:

Use the [Admin_RefreshSecurity] tag. This tag is required for new settings to go into effect if Lasso Security is manually altered outside of using Lasso Administration. This tag requires Lasso global administrator permission to use.

```
[Admin_RefreshSecurity]
```

Scheduling Events

Lasso includes a built-in scheduling facility that allows URL visits to be scheduled for a specific time in the future or to be scheduled for repeated visits. The scheduling facility loads the pages as if a client Web browser had visited the specified URL at the specified time.

Since the URLs can reference any format files available to Lasso Service, this simple scheduling facility allows powerful events to be scheduled that can perform any database actions or programming commands available to Lasso.

The scheduling facility can be used to schedule any of the following events.

- A **Routine Maintenance** page that performs database cleanup routines or optimizes the internal Lasso MySQL databases.
- A **Status Email** page that emails an administrator's address with information about Lasso's current status and what database actions have occurred since the last status email.
- A **Cache Update** page that performs a database search. The results of the search are stored and used instead of the full search in order to increase performance. The cached date is updated periodically.

- Events can load pages on **Remote Servers** in order to retrieve information, trigger an action on the remote server, or check that the remote server is active.

The event scheduling facility is intended for scheduling events which will be executed within about a minute of their intended execution time. It is not intended for high-precision execution of events. Please see the *Extending Lasso Guide* for information about how to create custom LDML tags that can execute with greater precision.

Event Administration

Lasso Administration allows events to be scheduled, the event queue to be stopped and started, and scheduled events to be viewed, modified, and deleted. See *Chapter 9: Administration Utilities* in the Lasso Professional 7 Setup Guide for more information.

Note: The event queue can be stopped in Lasso Administration, but will always be started again if Lasso Service is relaunched.

Event Tags

Events are scheduled using the [Event_Schedule] tag which is described in *Table 3: Scheduling Tags*.

Table 3: Scheduling Tag

Tag	Description
[Event_Schedule]	Schedules a URL to be loaded by Lasso at a specified time in the future or schedules repeated loads of a specified URL.

The [Event_Schedule] tag accepts many different parameters which are described in *Table 4: Scheduling Parameters*.

Table 4: Scheduling Parameters

Tag	Description
-URL	The URL which is to be loaded when the event executes. The URL can include URL parameters. Required.
-Start	The date/time to execute the event or the time to start executing a repeating event. Optional, defaults to the current time.
-End	The date/time to stop executing a repeating event. Optional, defaults to never.

-Delay	The number of minutes to wait between executions of a repeating event. Defaults to not repeating if no -Delay is specified.
-Restart	A boolean value specifying whether an event should continue execution after a restart. Defaults to True.
-Username	An optional username which will be used to authenticate the request for the event URL. Optional.
-Password	An optional password which will be used to authenticate the request for the event URL. Optional.

The parameters of the [Event_Schedule] tag interact to allow a great variety of different behaviors. The following examples make the use of the parameters clear.

To schedule an event to execute immediately:

Use the [Event_Schedule] tag with only a -URL parameter. The event will execute within about a minute of when it is scheduled. This allows a task specified in a separate format file to be executed separately from the main flow of the current format file.

```
[Event_Schedule: -URL='http://www.example.com/event.lasso']
```

To schedule an event to happen at a specific time:

Use the [Event_Schedule] tag with a -URL parameter and a -Start parameter. The event will execute within about a minute of the -Start time.

- The following event is scheduled to execute at midnight on April 5, 2005. Since the server will likely be restarted before that date, -Restart is set to True to ensure this event is not deleted when the server is next restarted.

```
[Event_Schedule:
  -URL='http://www.example.com/event.lasso',
  -Start='4/5/2005 00:00:00',
  -Restart=True]
```

- The following event is scheduled to execute at 4:00 PM on April 5, 2005. The date/time is specified in Lasso date format. -Restart is set to True to ensure this event is not deleted when the server is next restarted.

```
[Event_Schedule:
  -URL='http://www.example.com/event.lasso',
  -Start='4/5/2005 16:00:00'
  -Restart=True]
```

- The following event is scheduled to execute four hours after the page is loaded. The date/time for the -Start parameter is generated using the [Date_Add] and [Date_GetCurrentDate] tags.

```
[Event_Schedule:
  -URL='http://www.example.com/event.lasso',
  -Start=(Date_Add: (Date_GetCurrentDate), -Hour=4)]
```

To schedule an event to repeat:

Use the [Event_Schedule] tag with a -URL parameter, and a -Delay parameter which specifies that the event should repeat. -Restart is set to False. This event will not execute after the server is restarted unless we set -Restart to True.

- The following event is scheduled to repeat every fifteen minutes starting immediately.

```
[Event_Schedule:
  -URL='http://www.example.com/event.lasso',
  -Delay=15,
  -Restart=False]
```

- The following event is scheduled to repeat every hour on 12/25/2005. Note that -Restart is set to True so this event will not be deleted when Lasso Service is next restarted.

```
[Event_Schedule:
  -URL='http://www.example.com/event.lasso',
  -Start='12/25/2005 00:00:00',
  -End='12/26/2005 00:00:00',
  -Delay=60,
  -Restart=True]
```

To schedule an event with a complex schedule:

Events which need to execute on a complex schedule must be rescheduled every time they are executed. If an event is being rescheduled explicitly then the -Delay parameter should not be specified.

For example, the following code included in the `http://www.example.com/event.lasso` format file will reschedule the same format file as an event depending on whether or not the condition in the [If] ... [/If] tags is True. If the condition is True then the event will be rescheduled for fifteen minutes in the future, otherwise the event will be rescheduled for two hours in the future.

```
[If: (Variable: 'Reschedule') == True]
  [Event_Schedule:
    -URL='http://www.example.com/event.lasso',
    -Start=(Date_Add: (Date_GetCurrentDate), -Minute=15),
    -Restart=False]
[Else]
  [Event_Schedule:
```

```

-URL='http://www.example.com/event.lasso',
-Start=(Date_Add: (Date_GetCurrentDate), -Hour=2),
-Restart=False]
[/If]

```

Process Tags

The [Process] tag can be used to process LDML code which is contained within a variable or database field. The LDML code is processed as if it were contained in the current format file at the location of the [Process] tag. The code which is processed must be complete, all container tags must be closed within the processed code.

The [NoProcess] ... [/NoProcess] tag can be used to have Lasso ignore a portion of a page. Any LDML including square bracket or LassoScript syntax which is contained within the [NoProcess] ... [/NoProcess] tags will not be processed and will be passed through to the browser unchanged. This is most useful for segments of client-side JavaScript which contains array references using square brackets or to display a sample of LDML code on a page.

The [Sleep] tag can be used to pause the Lasso processing of the current format file for a specified number of milliseconds. This may be useful if Lasso actions need to be synchronized with the actions of other applications on the Web server or on other servers.

Table 5: Process Tags

Tag	Description
[Process]	Processes its parameter as LDML code.
[NoProcess] ... [/NoProcess]	Lasso will not process any LDML code contained within this container tag.
[Sleep]	Pauses execution of the current format file for a specified number of milliseconds.

Note: The [NoProcess] ... [/NoProcess] tags cannot be used within a LassoScript or within code processed by the [Process] tag. They must be typed exactly as specified here without any parameters or spaces within the square brackets in order to work.

To process code stored in a variable:

The following example shows how to store LDML code in a variable and then process it using the [Process] tag. The result is the same as if the code has been executed within the current format file at the location of the [Process] tag.

```
[Variable: 'LDML_Code'=(Server_Name)]
[Output: 'The current server is ']
[Process: (Variable: 'LDML_Code')]
[Output: '.']
```

→ The current server is www.example.com.

To process code stored in a database field:

The following example shows how to process code which is stored in a database variable. The result is the same as if the code has been executed within the current format file at the location of the [Process] tag. All records from the MyCode table of the Example database are found and the code from the Code field is executed.

```
[Inline: -Database='Example', -Table='MyCode', -FindAll]
[Records]
[Process: (Field: 'Code')]
[/Records]
[/Inline]
```

The result will be the result of the Code field for each record.

To instruct Lasso not to process a portion of a page:

Use the [NoProcess] ... [/NoProcess] tags. In the following HTML page none of the square brackets within the JavaScript will be processed by Lasso. This allows the JavaScript to be parsed properly by the browser without any additional work by the page developer.

```
<html>
<head>
  <title>My Lasso Page!</title>
  [NoProcess]
  <script language="JavaScript">
    ...
  </script>
[/NoProcess]
</head>
<body>
  ... Lasso code here will be processed ...
</body>
</html>
```

The [NoProcess] ... [/NoProcess] tags can also be used selectively around a small portion of a page that contains square brackets, but shouldn't be processed. For example, if you are using square brackets to decorate links you can use the [NoProcess] ... [/NoProcess] tags to ensure the contents of the square brackets is not processed.

```
<a href="mylink.lasso">[NoProcess][MyLink][NoProcess]</a>
```


→ `[MyLink]`

To pause execution of a format file for 15 seconds:

Use the `[Sleep]` tag with a parameter of 15000.

`[Sleep: 15000]`

Null Data Type

The null data type is the base type for all other data types in LDML. All of the tags of the null data type are available for use with values of any data type in LDML. Several of the member tags of the null data type such as `[Null->Type]` have already been introduced.

Table 6: Null Member Tags

Tag	Description
<code>[Null]</code>	Returns a null literal. This tag is usually used in comparisons or as a default value for new variables.
<code>[Null->DetachReference]</code>	Resets any reference value to null, detaching it from the master value and allowing it to be reassigned without affecting the master value. See the Extending Lasso Guide for information on references.
<code>[Null->Dump]</code>	Outputs all the variables and tags for a type. This is useful as a debugging tool.
<code>[Null->FreezeType]</code>	Freezes the type of a value so it cannot be modified. An error is thrown if a subsequent tag attempts to change the type of the value.
<code>[Null->FreezeValue]</code>	Freezes the value for a data type. An error is thrown if a subsequent tag attempts to modify the value.
<code>[Null->IsA]</code>	Requires a type name as a parameter. Returns true if the object is of that type or inherits from that type.
<code>[Null->Properties]</code>	Returns a pair containing two maps. The first element is a map of all member variables in the type. The second element is a map of all member variables in the type.
<code>[Null->RefCount]</code>	Returns the number of variables that reference the object.
<code>[Null->Serialize]</code>	Converts the value to a byte stream representation. The returned string can be stored in a database.
<code>[Null->UnSerialize]</code>	Accepts a single parameter which is a byte stream that represents a Lasso value. The current value is replaced by the value represented by the parameter.
<code>[Null->Type]</code>	Returns the data type of the value.

[Null->XMLSchemaType]	Returns the type for the root data type in the standard Lasso types schema.
-----------------------	---

To return the type of any variable:

Use the [Null->Type] tag. This tag returns a string which represents the data type of the value. If the data type is not defined then 'null' is returned. The following examples show the use of [Null->Type] on literals of different data types.

```
[Output: 123->Type] → Integer
[Output: 123.456->Type] → Decimal
[Output: 'String'->Type] → String
[Output: Null->Type] → Null
[Output: (Array: 1, 2, 3)->Type] → Array
```

To store a complex data type:

Use the [Null->Serialize] to transform the data type into a byte stream string representation that can be stored in a database field. Then use [Null->Unserialize] to transform the byte stream string representation back into the original data type. The following example shows how to convert an array into a string and then back again.

- 1 Store the array in a variable `ArrayVariable`.

```
[Variable: 'ArrayVariable'=(Array: 'one', 'two', 'three', 'four', 'five')]
```

- 2 Use the [Null->Serialize] tag to change the array into a string stored in `TempVariable`.

```
[Variable: 'TempVariable'=$ArrayVariable->Serialize]
```

- 3 The string representation of the array can now be changed back into the array by creating a new variable `ArrayVariable` and then calling the [Null->UnSerialize] tag with `TempVariable` as a parameter.

```
[Variable: 'ArrayVariable'=Null]
[$ArrayVariable->(UnSerialize: $TempVariable)]
```

- 4 Finally, the original array is output.

```
[Variable: 'ArrayVariable']
```

```
→ (Array: (one), (two), (three), (four), (five))
```

Extending Lasso Note: The null data type tags are used primarily to create custom tags and custom data types. To see more examples of null data type tag usage, see *Chapter 3: Custom Tags* and *Chapter 4: Custom Types* in the Extending Lasso Guide.

Page Variables

Lasso stores many internal values in variables which can be accessed by an LDML programmer. Lasso also provides a tag that allows access to all of the variables defined for a page as a map. These tools can be used by LDML programmers to perform low-level tasks that would not be possible otherwise.

Table 7: Page Variable Tags

Tag	Description
[Variables]	Returns a map containing every variable defined in a page. Can be abbreviated [Vars].
[Tags]	Returns a map containing every substitution, container, and process tag registered globally in Lasso. The map does not contain custom tags defined on the current page.

To list all variables defined in the current page:

Use the [Map->Keys] tag on the map returned by [Variables] to display the name of each variable defined on the current page.

```
[Variables->Keys]
```

→ (Array: (__html_reply__), (__result_code__), (__result_message__),
(__http_header__), (__tag_registry__))

To list all substitution, container, and process tags available in LDML:

Use the [Map->Keys] tag on the map returned by [Tags] to display the name of every tag defined globally within Lasso.

```
[Tags->Keys]
```

The output of this code is a list of close to four hundred tags registered globally in Lasso. Please see the tag list in **Appendix A: LDML 7 Tag List** for a listing of the standard substitution tags.

The list of all custom tags defined on the current page can be returned using the following code. All custom tags defined on the current page are stored in the __tags__ variable.

```
[Output: (Variable: '__tags__')->Keys]
```

The format will be the same as that for the [Tags] map above.

Table 8: Page Variables

Variable	Description
<code>__encoding__</code>	The character set which will be used to encode the page for delivery to the Web client. Set by [Content_Type].
<code>__html_reply__</code>	The current output of the format file is stored in this variable. Changing this variable will alter the output that will be sent to the site visitor.
<code>__http_header__</code>	The header that will be returned by Lasso as part of the http response. The header can be manipulated using the [Header] ... [/Header] tags.
<code>__tags__</code>	Contains a map of all custom tags defined in the current page. The global version of this variable is accessible using the [Tags] tag documented above.
<code>__prototypes__</code>	A map of all default member tags for each LDML 7 data type is stored in this global variable (see the Extending Language Guide for more information).

Warning: Altering any variables whose names start with an underscore is not recommended except using the methods documented in this section. These variables store internal values that are used by Lasso while processing LDML.

To alter the output of the current format file:

Set the variable `__html_reply__` to the desired output. In the following example, an access denied message is displayed to the user rather than the output of the current format file.

```
<?LassoScript
    // This script changes the output of the page to an access denied message.

    $__html_reply__ = '<html>\n';
    $__html_reply__ += '\t<head><title>Access Denied</title></head>\n';
    $__html_reply__ += '\t<body><h1>Access Denied</h1></body>\n';
    $__html_reply__ += '</html>\n';
?>
```

→ `<html>`
`<head><title>Access Denied</title></head>`
`<body><h1>Access Denied</h1></body>`
`</html>`

Any LDML tags or HTML code in the format file after this LassoScript will be appended to the end of the output variable. An [Abort] tag can be used to halt execution of the page and output the contents of the variable immediately.

Configuration Tags

Lasso provides a number of tags that allow the current configuration to be examined. These tags are summarized in *Table 9: Configuration Tags*.

Table 9: Configuration Tags

Tag	Description
[Lasso_DatasourceIsFileMaker]	Accepts the name of a single database. Returns True if the database is being served through Lasso Connector for FileMaker Pro.
[Lasso_DatasourceIsLassoMySQL]	Accepts the name of a single database. Returns True if the database is being served through Lasso Connector for Lasso MySQL.
[Lasso_DatasourceIsMySQL]	Accepts the name of a single database. Returns True if the database is being served through Lasso Connector for MySQL.
[Lasso_DatasourceModuleName]	Accepts the name of a single database. Returns the name of the data source connector for the database.
[Lasso_TagExists]	Checks to see if a substitution or process tag is defined. Returns True or False. Requires one parameter which is the name of the tag to be checked.
[Lasso_TagModuleName]	Returns the name of the module in which a tag is defined. Requires one parameter which is the name of the tag to be checked.
[Lasso_Version]	Returns the version of Lasso Professional.

To check whether a tag exists:

Use the [Lasso_TagExists] tag with the tag name of the substitution or process tag to be checked. The following example will return True if the [Email_Send] tag is defined.

```
[Lasso_TagExists: 'Email_Send']
```

→ True

To check what module a tag is defined in:

Use the [Lasso_TagModuleName] tag with the name of the substitution or process tag to be checked. The following example will return the module that defines the [NSLookup] tag, NSLookup.class.

```
[Lasso_TagModuleName: 'NSLookup']
```

→ NSLookup.class

Format File Execution Time Limit

Lasso includes a limit on the length of time that a format file will be allowed to execute. This limit can help prevent errors or crashes caused by infinite loops or other common coding mistakes. If a format file runs for longer than the time limit then it is killed and a critical error is returned and logged.

The execution time limit is set to 10 minutes (600 seconds) by default and can be modified or turned off in the *Setup > Global > Settings* section of Lasso Admin. The execution time limit cannot be set below 60 seconds.

Table 10: Time Limit Tags

Tag	Description
[Lasso_ExecutionTimeLimit]	Sets the time limit for an individual format file.

The limit can be overridden on a case by case basis by including the [Lasso_ExecutionTimeLimit] tag at the top of a format file. This tag can set the time limit higher or lower for the current page allowing it to exceed the default time limit. Using [Lasso_ExecutionTimeLimit: 0] will deactivate the time limit for the current format file altogether.

On servers where the time limit should be strictly enforced, access to the [Lasso_ExecutionTimeLimit] tag can be restricted in the *Setup > Global > Tags* and *Security > Groups > Tags* sections of Lasso Admin.

Asynchronous tags and compound expressions are not affected by the execution time limit. These processes run in a separate thread from the main format file execution. If a time limit is desired in an asynchronous tag the [Lasso_ExecutionTimeLimit] tag can be used to set one.

Note: When the execution time limit is exceeded the thread that is processing the current format file will be killed. If there are any outstanding database requests or network connections open there is a potential for some memory to be leaked. The offending page should be reprogrammed to run faster or exempted from the time limit using [Lasso_ExecutionTimeLimit: 0]. Restarting Lasso Service will reclaim any lost memory.

23

Chapter 23

Miscellaneous Tags

This chapter documents several tags which do not logically fit into any other chapter in the Lasso 7 Language Guide.

- *Name Server Lookup* documents the [NSLookup] tag.
- *Validation Tags* describes tags which validate credit card numbers, email addresses, and URLs.
- *Unique ID Tags* describes the [Lasso_UniqueID] tag.

Name Server Lookup

The [NSLookup] tag is implemented using LJAPI (Lasso Java API). In order to use the [NSLookup] tag, Java must be configured properly. Please see *Chapter 6: Setting Global Preferences* and the configuration chapters of the Lasso Professional 7 Setup Guide for more information.

Table 1: Name Server Lookup Tag

Tag	Description
[NSLookup]	Requires a single parameter. Returns the IP address if the parameter is a host name or the host name if the parameter is an IP address.

To find the IP address of a specific host name:

Use the [NSLookup] tag with the host name as its parameter. The following example returns the IP address for `www.example.com`.

[NSLookup: 'www.example.com'] → 127.0.0.1

To find the host name for a specific IP address:

Use the [NSLookup] tag with the IP address as its parameter. The following example returns the host name for the IP address 127.0.0.1.

[NSLookup: '127.0.0.1'] → www.example.com

Validation Tags

LDML provides a set of tags which can be used to validate various text formats. These tags are summarized in *Table 2: Valid Tags*.

Table 2: Valid Tags

Tag	Description
[Valid_CreditCard]	Accepts a single string parameter containing a credit card number. Returns True if the credit card number is valid according to the ROT-13 algorithm.
[Valid_Date]	Accepts a single string parameter containing a date. Returns True if the date is in a format that Lasso can parse or False otherwise.
[Valid_Email]	Accepts a single string parameter containing an email address. Returns True if the email address appears to be in a valid format.
[Valid_URL]	Accepts a single string parameter containing a URL. Returns True if the URL appears to be in a valid format.

Note: See *Chapter 14: String Operations* for information about the [String_Is...] tags that can be used to determine what type of data strings contain.

To check whether a credit card number is valid:

The [Valid_CreditCard] tag provides a quick check for the basic validity of a credit card number, but can only ensure that a card is of the right format, not that an account is active or has available credit. The following code checks the fake credit card number 8888 8888 8888 8888 and predictably returns False.

[Valid_CreditCard: '8888888888888888'] → False

Unique ID Tags

The [Lasso_UniqueID] tag can be used to create a simple unique ID. The ID created by the [Lasso_UniqueID] tag has a very high probability of being unique since it is based on the current date and time, the IP address of the current visitor, and a random component.

Unique IDs are usually used to identify a particular record in a database. When a new record is added, one field is set to the value from [Lasso_UniqueID] and that same value is stored in a variable. When the record needs to be retrieved from the database, [Lasso_UniqueID] can be used again.

Table 3: Unique ID Tag

Tag	Description
[Lasso_UniqueID]	Returns a unique ID.

Server Tags

The following tags provide useful information when logging to the console. The type of output can be selected by specifying an optional parameter.

Table 4: Server Tags

Tag	Description
[Server_Date]	Returns the current date. Accepts a parameter -Short, -ShortY2K, -Abbrev, or -Long.
[Server_Day]	Returns the current weekday. Accepts a parameter -Short or -Long.
[Server_Time]	Returns the current time. Accepts a parameter -Short, -Long, -Extended.

24

Chapter 24

LassoScript

LassoScript is an alternate method of specifying LDML code. This chapter describes the syntax rules and conventions of LassoScript.

- *LassoScript Overview* describes LassoScript in general and explains when its use is preferred.
- *LassoScript Syntax* fully documents the syntax rules and delimiters for LassoScript.

LassoScript Overview

LassoScript allows LDML tags and symbols to be used as a scripting language in a fashion which is complementary with the traditional use of LDML as a tag-based markup language. LassoScripts have the following advantages:

- Concise format allows better formatting of long code segments.
- Represented as a single object in many visual authoring environments. This makes it easy to separate the logic of a page from the presentation or to hide implementation details from Web designers who are working on the visual aspects of the page.
- Comments allow code to be self-documented for better maintainability.
- Compatible with HTML and XML standards for embedding server-side scripting commands.
- Provides scripting-like method of coding for programmers who prefer this method.

LDML tags contained within a LassoScript execute exactly as they would if they were specified within square brackets. The value returned by a

LassoScript is the concatenation of all the values which are returned from the tags that make up the LassoScript. No encoding is applied to the output of a LassoScript, but normal encoding rules apply to each of the tags within a LassoScript that outputs values.

The [Output] tag is used to determine the output of values from a LassoScript and to apply explicit encoding to values output from a LassoScript.

```
<?LassoScript
  Output: '<br>This is the output from the LassoScript.', -EncodeNone;
?>
```

→
This is the output from the LassoScript.

LassoScript Syntax

LassoScript's begin with <?LassoScript and end with ?>. LDML tags within a LassoScript are delimited by a single semi-colon ; at the end of the tag rather than by square brackets. White space within a LassoScript is ignored. Comments begin with a double forward slash // and continue to the end of the line. To continue a comment on another line, another // must be used. All text in a LassoScript must be part of a tag or part of a comment, no extraneous text is allowed.

Table 1: LassoScript Delimiters

Delimiter	Description
<?LassoScript	Starts a LassoScript. Required.
?>	Ends a LassoScript. Required.
;	Ends an LDML tag. Required.
//	Comment. All text to the end of the line will be ignored.
/* ... */	Block Comment. All text between the delimiters will be ignored. Allows multi-line comments.

Note: Square brackets [...] are not allowed in LassoScripts. Use parentheses instead to group tag names and their parameters (Tag_Name: -Parameter);.

To create a LassoScript with a single tag:

LassoScripts can be used for individual LDML tags as well as for groups of tags. When only a single tag is specified, the semi-colon at the end of the tag is optional. The container <?LassoScript ... ?> can be substituted for the square bracket container [...] if necessary.

```
<?LassoScript Field: 'Field_Name', -EncodeNone ?>
```

```
<?LassoScript Math_Add: 1, 2, 3, 4, 5 ?> → 15
```

To use container tags within a LassoScript:

Container tags are specified just as they are in square-bracketed LDML. The opening container tag must end with a semi-colon. The closing container tag should start with a forward slash / and end with a semi-colon. Indentation is usually used to make the contents of the container tag clear, but is not required. In the following example a [Loop] tag is used to output the numbers 1 through 5 using the [Loop_Count] tag.

```
<?LassoScript
  Loop: 5;
  Output: (Loop_Count) + ' ';
/Loop;
?>
```

```
→ 1 2 3 4 5
```

To use container tags between LassoScripts:

Container tags can be opened within one LassoScript then closed in a subsequent LassoScript. The following example shows a mixture of LassoScripts and square bracket syntax which implements a loop.

```
<?LassoScript
  Loop: 5;
?>

  [Output: (Loop_Count) + ' ']

<?LassoScript
  /Loop;
?>
```

```
→ 1 2 3 4 5
```

To specify a comment within a LassoScript:

Use the // symbol to start a comment. All text until the end of the line will be part of the comment and will not be executed by the LassoScript.

```
<?LassoScript
  // This LassoScript only contains a comment.
?>

<?LassoScript
  // The following line has been commented out. It will not be processed.
  // Output: 'Testing';
?>
```

Alternately, specify a multi-line comment using the `/* ... */` delimiter. All text between these delimiters will not be processed.

```
<?LassoScript
/*
    These lines have been commented out. The following line will not be processed.
    Output: 'Testing';
*/
?>
```

To suppress output from a LassoScript:

Use the `[Output_None]` ... `[/Output_None]` tags around the LassoScript. In the following example, the LassoScript will return no value even though it contains several `[Output]` tags.

```
<?LassoScript
    Output_None;

    Output: 'This value will not be seen.';
    Output: 'Neither will this value.';

    /Output_None;
?>
```

To change the encoding for a LassoScript:

Use the `[Encode_Set]` ... `[/Encode_Set]` tags around the LassoScript. In the following example, the LassoScript will not perform any encoding so HTML values can output from the `[Output]` tags without use of the `-EncodeNone` keyword in each tag.

```
<?LassoScript
    Encode_Set: -EncodeNone;

    Output: '<p>This HTML code will render<br>with breaks.';

    /Encode_Set;
?>
```

→ `<p>This HTML code will render

with breaks.`

To convert LDML square bracket code to a LassoScript:

- 1 Format the code so each tag is on a separate line.
- 2 Remove all opening square brackets `[`.
- 3 Replace all closing square brackets `]` with semi-colons `;`.
- 4 Correct the indentation so tags inside container tags are indented.

5 Add `<?LassoScript` and `?>` to the beginning and end of the code.

In the following example the same code is shown in square bracketed LDML code and then as an equivalent LassoScript.

```
[Loop: 5]
  [Output: (Loop_Count) + ' ']
[/Loop]

<?LassoScript
  Loop: 5;
  Output: (Loop_Count) + ' ';
/Loop;
?>
```


IV

Section IV

Protocols and Media

This section contains information about tags and techniques for interacting with remote Web servers, sending email, serving multimedia files, supporting WML and XML browsers and more.

- *Chapter 25: Email* contains information about how to send email from Lasso.
- *Chapter 26: Images and Multimedia* contains information about how to serve images and multimedia files from Lasso.
- *Chapter 27: HTTP/HTML Content and Controls* describes tags that allow the information in the HTTP request to be inspected and for the HTTP response to be customized.
- *Chapter 28: Wireless Devices* contains information about how to serve WML pages to WAP equipped devices such as personal digital assistants (PDAs) and cell phones.
- *Chapter 29: XML* includes information about how to serve XML data to clients such as other Web application servers or Web browsers that support XML and how to send and receive XML-RPC requests.
- *Chapter 30: PDF* includes information about how to create dynamic PDF documents and serve them.

25

Chapter 25

Email

This chapter describes how to send email using Lasso.

- *Sending Email* describes the [Email_Send] tag.

Sending Email

Lasso includes a built-in system for queuing and sending email messages from LDML format files. Email messages can be sent to site visitors to notify them when they create a new account or to remind them of their login information. Email messages can be sent to administrators when various errors or other conditions occur. Email messages can even be sent in bulk to many email addresses to notify site visitors of updates to the Web site or other news.

Email messages are queued using the [Email_Send] tag. Lasso's email system checks the queue periodically and sends any messages which have been queued. If the email system encounters an error when sending an email then it requeues the message.

Lasso sends each message to the SMTP server specified in the [Email_Send] tag or in Lasso Administration. Lasso must have permission to send email through an SMTP server in order to use the [Email_Send] tag. Consult the documentation for your SMTP server in order to ensure that the machine which is hosting Lasso Service has permission to relay mail to your SMTP server.

The email system is administered using the *Monitor > Email* section of Lasso Administration. The *Email Queue* can be inspected and any errors which have occurred can be reviewed. Email messages can be queued manually using the *Send Email* page. The preferences for the email system,

such as how often the queue is checked for messages or how many times messages are requeued if an error is detected, can be modified using the *Setup* page. See *Chapter 9: Administration Utilities* of the Lasso Professional 7 Setup Guide for more information.

Note: Lasso’s email system is written in LDML using the TCP/IP tags. See the *Extending Lasso Guide* for information about how to create a new solution based on the email system.

Table 1: Email Tag

Tag	Description
[Email_Send]	Queues an email message.

The [Email_Send] tag accepts many keyword/value parameters which are detailed in *Table 2: [Email_Send] Parameters*.

Table 2: [Email_Send] Parameters

Keyword	Description
-Host	Optional SMTP host through which to send messages. Default is the host defined in Lasso Administration.
-To	The recipient of the message. Multiple recipients can be specified by separating their email addresses with commas. At least one of -To, -CC, or -BCC is required.
-From	The sender of the message. The email address specified must have permission to send messages through the SMTP host. Required.
-Subject	The subject of the message. Required.
-Body	The body of the message. Required.
-CC	Carbon copy recipients of the message. Multiple recipients can be specified by separating their email addresses with commas. At least one of -To, -CC, or -BCC is required.
-BCC	Blind carbon copy recipients of the message. Multiple recipients can be specified by separating their email addresses with commas. At least one of -To, -CC, or -BCC is required.
-ExtraMIMEHeaders	A pair array which defines extra MIME headers that should be added to the email message. Optional.
-Attachments	An array of paths to files (or the path to a single file) which should be included as attachments with the message.

Note: The -Email... command tags from LDML 3 will still operate in LDML 7 for support of legacy solutions. These command tags are deprecated which means that they may not be available in a future release. The [Email_Send] tag should be used for all format files created using LDML 7.

To send an email message:

Use the [Email_Send] tag with the desired parameters. The -Host parameter must be set to a valid SMTP host. The -From parameter must be set to an email address which has permission to send messages through the SMTP host. The -Subject parameter must be set to the desired subject for the email message. One or more recipients must be specified using the -To, -CC, and -BCC parameters. The body of the message can be specified using any of the three methods described here.

- An email can be sent with a hard-coded body by specifying the message directly within the [Email_Send] tag. The following example shows an email sent to example@example.com with a hard-coded message body.

```
[Email_Send: -Host='mail.example.com',
  -To='example@example.com',
  -From='example@example.com',
  -Subject='An Email',
  -Body='This is the body of the email.']
```

- The body of an email message can be assembled in a variable in the current format file and then sent using the [Email_Send] tag. The following example shows a variable Email_Body which has several items added to it before the message is finally sent.

```
<?LassoScript
  Variable: 'Email_Body' = 'This is the body of the email';
  $Email_Body += "\nSent on: " + (Server_Date) + ' at ' + (Server_Time);
  $Email_Body += "\nCurrent visitor: " + (Client_Username) + ' at ' + (Client_IP);

  Email_Send: -Host='mail.example.com',
    -To='example@example.com',
    -From='example@example.com',
    -Subject='An Email',
    -Body=$Email_Body;
?>
```

- A format file on the Web server can be used as the message body for an email message using the [Include] tag. A format file created to be a message body should contain no extra white space. The following example shows a format file format.lasso which is contained in the same folder as the current format file being used as the message body for an

email. Any LDML tags within `format.lasso` will be executed before the email is sent.

```
[Email_Send: -Host='mail.example.com',
  -To='example@example.com',
  -From='example@example.com',
  -Subject='An Email',
  -Body=(Include: 'format.lasso')]
```

To send an email message to multiple recipients:

Email can be sent to multiple recipients by including their addresses as a comma delimited list in the `-To` parameter, the `-CC` parameter, or the `-BCC` parameter. Multiple `-To`, `-CC`, or `-BCC` parameters are not allowed.

- The following example shows an `[Email_Send]` tag with two recipients in the `-To` parameter. The recipients email addresses are specified with a comma between them `example@example.com`, `someone@example.com`. No extraneous information such as the recipients real names should be included.

```
[Email_Send: -Host='mail.example.com',
  -To='example@example.com, someone@example.com',
  -From='example@example.com',
  -Subject='An Email',
  -Body=(Include: 'format.lasso')]
```

- The following example shows an `[Email_Send]` tag with one recipient in the `-To` parameter and two recipients in the `-CC` parameter. The Carbon Copy parameter is generally used to include recipients who are not the primary recipient of the email, but need to be informed of the correspondence. The addresses for the carbon copied recipients are stored in variables and concatenated together with a comma between them using a `+` symbol.

```
[Variable: 'President'='president@example.com']
[Variable: 'Someone'='someone@example.com']
[Email_Send: -Host='mail.example.com',
  -To='example@example.com',
  -CC=($President + ',' + $Someone),
  -From='example@example.com',
  -Subject='An Email',
  -Body=(Include: 'format.lasso')]
```

- The following example shows an `[Email_Send]` tag with one recipient in the `-To` parameter and two recipients in the `-BCC` parameter. The Blind Carbon Copy parameter can be used to send email to many recipients without disclosing the full list of recipients to everyone who receives the

email. Each recipient will receive an email that contains only the address in the -To parameter `announce@example.com`.

```
[Email_Send: -Host='mail.example.com',
-To='announce@example.com',
-BCC='example@example.com, someone@example.com',
-From='example@example.com',
-Subject='An Email',
-Body=(Include: 'format.lasso')]
```

To send HTML email:

HTML email requires a few extra MIME headers in order to inform the email client how to render the message. These MIME headers can be added to any outgoing email using the -ExtraMIMEHeaders keyword/value parameter. The following three parameters are required to send HTML email.

```
MIME-Version: 1.0
Content-Type: text/html
Content-Transfer-Encoding: 8bit
```

The following examples shows how to set up these three MIME headers by storing them in an array and then referencing the array in the [Email_Send] tag.

```
[Variable: 'MIME_Headers' = (Array: 'MIME-Version'='1.0',
'Content-Type'='text/html',
'Content-Transfer-Encoding'='8bit')]

[Email_Send: -Host='mail.example.com',
-To='example@example.com',
-From='example@example.com',
-Subject='An HTML Email',
-Body='<b>This is the body of an HTML Email.</b>',
-ExtraMIMEHeaders=$MIME_Headers]
```

To send an email message with a Reply-To header:

Any valid email headers can be included with -ExtraMIMEHeaders. For example, a Reply-To header will specify that a different address than the From address should be used for responses to a message.

```
Reply-To: someone@example.com
```

The following examples shows how to set up an array that contains a Reply-To header and then referencing the array in the [Email_Send] tag.

```
[Variable: 'MIME_Headers' = (Array: 'Reply-To'='someone@example.com')]
```

```
[Email_Send: -Host='mail.example.com',  
-To='example@example.com',  
-From='example@example.com',  
-Subject='AnEmail',  
-Body='This is the body of the Email.',  
-ExtraMIMEHeaders=$MIME_Headers]
```

To send attachments with an email message:

Files can be included as attachments to email messages using the -Attachments parameter. This parameter takes an array of file paths as a value. When the email is sent, each file is read from disk and encoded using Base-64 encoding. The recipient's email client will automatically decode the attached files and make them available.

The following example shows a single attachment being sent with an email message. The attachments are named MyAttachment.txt and MyAttachment2.txt. They are located in the same folder as the format file which is sending the email.

```
[Email_Send: -Host='mail.example.com',  
-To='example@example.com',  
-From='example@example.com',  
-Subject='AnEmail',  
-Body='This is the body of the Email.',  
-Attachments=(Array: 'MyAttachment.txt', 'MyAttachment2.txt')]
```


26

Chapter 26

Images and Multimedia

This chapter describes the methods which can be used to manipulate and serve images and multimedia files using Lasso.

- *Overview* provides an overview of the image manipulation and multimedia features included in Lasso Professional 7.
- *Casting Images as LDML Objects* describes how to cast image files as LDML objects so they can be dynamically edited using LDML.
- *Getting Image Information* describes how to access the attributes of an image using LDML.
- *Converting and Saving Images* describes how to convert images from one format to another, and how to save images to file using LDML.
- *Manipulating Images* describes how to edit image files using LDML.
- *Extended ImageMagick Commands* describes how to invoke extended ImageMagick functionality using LDML.
- *Serving Images and Multimedia Files* describes how to serve images and multimedia files through Lasso format files, and how to reference images and multimedia files stored within the Web server root.

Overview

Lasso Professional 7 includes features that allow you to manipulate and serve images and multimedia files on the fly. New LDML [Image] tags allow you to do the following with image files in supported image formats:

- Scaling and cropping images, facilitating the creation of thumbnail images on the fly.
- Rotating images and changing image orientation.

- Apply image effects such as modulation, blurring, and sharpening effects.
- Adjusting image color depth and opacity.
- Combining images, adding logos and watermarks.
- Image format conversion.
- Retrieval of image attributes, such as image dimensions, bit depth, and format.
- Executing extended ImageMagick commands.

Implementation Note: The image tags and features in LDML 7 are implemented using ImageMagick 5.5.7 (July 2003 build), which is installed as part of Lasso Professional 7 on Mac OS X 10.3. Windows requires ImageMagick to be installed separately, which is covered in chapter 4 of the Lasso Professional 7 Setup Guide. For more information on ImageMagick, visit <http://www.imagemagick.com>.

Introduction to Manipulating Image Files

Image files can be manipulated via LDML by setting a variable that references an image file on the server using the [Image] tag, and then using various member tags to manipulate the variable. Once the image file is manipulated, it can either be served directly to the client browser, or it can be saved to disk on the Web server.

To dynamically manipulate an image on the server:

The following shows an example of initializing, manipulating, and serving an image file named image.jpg using the [Image] tags.

```
[Var:'MyImage' = (Image: '/images/image.tif')]
[$MyImage->(Scale: -Height=35, -Width=35, -Thumbnail)]
[$MyImage->(Save: '/images/image.jpg')]


```

In the example above, an image file named image.tif is cast as a Lasso image data type using the [Image] tag, then resized to 35 x 35 pixels using the [Image->Scale] tag (the optional -Thumbnail parameter optimizes the image for the Web). Then, the image is converted to JPEG format and saved to file using the [Image->Save] tag, and displayed on the current page using an HTML tag.

This chapter explains in detail how these and other tags are used to manipulate image and multimedia files. This chapter also shows how to output an image file to a client browser within the context of a format file.

Supported Image Formats

Because the [Image] tags are based on ImageMagick, Lasso Professional 7 supports reading and manipulating over 88 major file formats (not including sub-formats). A comprehensive list of supported image formats can be found at the following URL.

<http://www.imagemagick.com/www/formats.html>

A list of commonly used image formats that are certified to work with Lasso Professional 7 out of the box without additional components installed are shown in *Table 1: Tested and Certified Image Formats*.

Table 1: Tested and Certified Image Formats

Format	Description
BMP	Microsoft Windows bitmap.
CMYK	Raw cyan, magenta, yellow, and black samples.
GIF	CompuServe Graphics Interchange Format. 8-bit RGB PseudoColor with up to 256 palette entries.
JPEG	Joint Photographic Experts Group JFIF format. Also known as JPG.
PNG	Portable Network Graphics.
PSD	Adobe Photoshop bitmap file.
RGB	Raw red, green, and blue samples.
TIFF	Tagged Image File Format. Also known as TIF.

Note: Many of the supported formats listed on the ImageMagick site such as EPS and PDF may be used with the [Image_...] tags, but require additional components such as Ghostscript to be installed before they will work. These formats may be used, but because they rely heavily on third-party components, they are not officially supported by Blue World.

File Permissions

This section describes the file permission requirements for manipulating files on a Web server using LDML 7. In order to successfully manipulate and save image files, the following conditions must be met.

- When saving image files using the [Image] tags, the user (e.g. AnyUser group for anonymous users) must have Create Files, Read Files, and Write Files permissions allowed in the *Setup > Security > Files* section of Lasso Administration, and the folder in which the image will be created must be available to the user within the Allow Path field.

- When creating files, Lasso Service (i.e. the Lasso user in Mac OS X or LocalSystem user in Windows) must be allowed by the operating system to write and execute files inside the folder. To check folder permissions in Windows, right-click on the folder and select *Properties > Security*. For Mac OS X, refer to the included *Mac_OS_X_Tips.pdf* document for instructions on changing file and folder permissions.
- Any file extensions being used by the [Image] tags must be allowed in the *Setup > Global Settings > Settings* section of Lasso Administration. This can include .gif, .jpg, .png, or any other supported image format you are using.

Casting Images as LDML Objects

For Lasso to be able to edit an image via LDML, an image file or image data must first be cast as an LDML image variable using the [Image] tag. Once a variable has been set to an image data type, various Image member tags can be used to manipulate the image. Once the image file is manipulated, it can either be served directly to the client browser, or it can be saved to disk on the Web server.

Table 2: [Image] Tag:

Tag	Description
[Image]	Casts an image as an LDML object. Requires either the name and path of an image file or a binary data string to initialize an image. Once an image is cast as an object, it may be edited and saved using [Image] member tags, which are described throughout this chapter.

Table 3: [Image] Tag Parameters:

Parameter	Description
'File Path'	Path to image file on the server. Required if -Binary or -Base64 is not specified.
-Binary	Creates an image file from binary image data. Requires a valid binary string for a supported image format. Required if a file path is not specified.
-Info	Optional parameter retrieves all the attributes of an image without reading the pixel data. Allows for better performance and less memory usage when casting an image (recommended for larger files).

To cast an image file as an LDML object:

Use the [Image] to initialize an image file so it can be manipulated by Lasso.

```
[Var:'MyImage1'=(Image: '/images/image.jpg')]
```

To cast a large image file as an LDML object:

Use the [Image] to initialize an image file using the -Info parameter for increased performance with larger files.

```
[Var:'MyImage2'=(Image: '/images/largeimage.jpg', -Info)]
```

To initialize an image from binary image data:

Lasso can create an image from a binary string for a valid image type using the [Image] tag with the -Binary parameter. The image is initialized and created in memory only until it is saved using the [Image->Save] tag described later.

```
[Var:'Binary'=(Include_Raw: 'image.jpg')]
[Var:'MyImage3'=(Image: -Binary=$Binary)]
```

Getting Image Information

Information about an image can be returned using special [Image] member tags. These tags return specific values representing the attributes of an image such as size, resolution, format, and file comments. All image information tags in LDML 7 are defined in *Table 4: Image Information Tags*.

Table 4: Image Information Tags

Tag	Description
[Image->Width]	Returns the image width in pixels. Integer value returned.
[Image->Height]	Returns the image width in pixels. Integer value returned.
[Image->ResolutionH]	Returns the horizontal resolution of the image in dpi. Integer value returned.
[Image->ResolutionV]	Returns the vertical resolution of the image in dpi. Integer value returned.
[Image->Depth]	Returns the color depth of the image in bits. Can be either 8 or 16.
[Image->Format]	Returns the image format (GIF, JPEG, etc).

[Image->Pixel]	Returns the color of the pixel located at the specified pixel coordinates (X,Y). The returned value is an array of RGB color integers (0-255) by default. An optional -Hex parameter returns a hex color string (#FFCCDD) instead of an RGB array.
[Image->Comments]	Returns any comments included in the image file header.
[Image->Describe]	Lists various image attributes, mostly for debugging purposes. An optional -Short parameter displays abbreviated information.
[Image->File]	Returns the image file path and name, or null for in-memory images.

To return the height and width of an image:

Use the [Image->Height] and [Image->Width] tags on a defined image variable. This returns an integer value representing the height and width of the image in pixels.

```
[Var: 'MyImage'=(Image: '/images/image.jpg')]
[$MyImage->Width] x [$MyImage->Height]
```

→ 400 x 300

To return the resolution of an image:

Use the [Image->ResolutionH] and [Image->ResolutionV] tags on a defined image variable. This returns a decimal value representing the horizontal and vertical dpi (dots per inch) of the image.

```
[Var: 'MyImage'=(Image: '/images/image.jpg')]
[$MyImage->ResolutionV] x [$MyImage->ResolutionH]
```

→ 600 x 600

To return the color depth of an image:

Use the [Image->Depth] tag on a defined image variable. This returns an integer value representing the color depth of an image in bits.

```
[Var: 'MyImage'=(Image: '/images/image.jpg')]
[$MyImage->Depth]
```

→ 16

To return the format of an image:

Use the [Image->Format] tag on a defined image variable. This returns a string value representing the file format of the image.

```
[Var: 'MyImage'=(Image: '/images/image.gif')]
[$MyImage->Format]
```

→ GIF

To return pixel information about an image:

Use the [Image->Pixel] tag on a defined image variable. This returns a string value representing the color of the pixel at the specified coordinates.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Pixel: 25, 125, -Hex)]
```

→ FF00FF

Converting and Saving Images

This section describes how image files can be converted from one format to another and saved to file. This is all accomplished using the [Image->Save] tag, which is described in the following table.

Table 5: Image Conversion and File Tags

Tag	Description
[Image->Convert]	Converts an image variable to a new format. Requires a file extension as a string parameter which represents the new format the image is being converted to (e.g. 'jpg', 'gif'). A -Quality parameter specifies the image compression ratio (integer value of 1-100) used when saving to JPEG or GIF format.
[Image->Save]	Saves the image to a file in a format defined by the file extension. Automatically converts images when the extension of the image to save as differs from that of the original image. A -Quality parameter specifies the image compression ratio (integer value of 1-100) used when saving to JPEG or GIF format.
[Image->AddComment]	Adds a file header comment to the image before it is saved. Passing no parameters removes any existing comments.

To convert an image file from one format to another:

Use the [Image->Convert] and [Image->Save] tags on a defined image variable, specifying the new format as part of the [Image->Convert] tag.

```
[Var: 'MyImage' =(Image: '/images/image.gif')]
[$MyImage->(Convert: 'JPG', -Quality=100)]
[$MyImage->(Save: '/images/image.jpg', -Quality=100)]
```

To automatically convert an image file from one format to another:

Use the [Image->Save] tag on a defined image variable, changing the image file extension to the desired image format. A -Quality parameter value of 100 specifies that the resulting JPEG file will be saved at the highest-quality resolution.

```
[Var: 'MyImage' =(Image: '/images/image.gif')]
[$MyImage->(Save: '/images/image.jpg', -Quality=100)]
```

To save a defined image variable to file:

Use the [Image->Save] tag on a defined image variable, specifying the desired image name, path, and format.

```
[Var: 'MyImage' =(Image: '/folder/asdf1.jpg')]
[$MyImage->(Save: '/images/image.jpg')]
```

To rename an image:

Use the [Image->Save] tag on a defined image variable, changing the existing image file name to the desired image file name.

```
[Var: 'MyImage' =(Image: '/images/image.gif')]
[$MyImage->(Save: '/images/image.jpg', -Quality=100)]
```

To add a comment to an image file header:

Use the [Image->AddComment] tag to add a comment to a defined image variable before it is saved to file. This comment is not displayed, but stored with the image file information.

```
[Var: 'MyImage' =(Image: '/images/image.gif')]
[$MyImage->(AddComment: 'This is a comment')]
[$MyImage->(Save: '/images/image.gif')]
```

Manipulating Images

Images can be transformed and manipulated using special [Image] member tags. These tags change the appearance of the image as it served to the client browser. This includes tags for changing image size and orientation, applying image effects, adding text to images, and merging images, which are described in the following sub-sections.

Changing Image Size and Orientation

Lasso provides tags that allow you to scale, rotate, crop, and invert images. These tags are defined in *Table 6: Image Size and Orientation Tags*.

Table 6: Image Size and Orientation Tags

Tag	Description
[Image->Scale]	Scales an image to a specified size. Requires -Width and -Height parameters, which specify the new size of the image using either integer pixel values (e.g. 50) or string percentage values (e.g. '50%'). An optional -Sample parameter indicates high-quality sampling should be used. An optional -Thumbnail parameter optimizes the image for display on the Web.
[Image->Rotate]	Rotates an image clockwise by the specified amount in degrees (integer value of 0-360). An optional -BGColor parameter specifies the hex color to fill the blank areas of the resulting image.
[Image->Crop]	Crops the original image by cutting off extra pixels beyond the boundaries specified by the parameters. Requires -Height and -Width parameters which specify the pixel size of the resulting image, and -Left and -Right parameters specify the offset of the resulting image within the initial image.
[Image->FlipV]	Creates a vertical mirror image by reflecting the pixels around the central X-axis.
[Image->FlipH]	Creates a horizontal mirror image by reflecting the pixels around the central Y-axis.

To enlarge an image:

Use the [Image->Scale] tag on a defined image variable. The following example enlarges image.jpg to 225 X 225 pixels. The optional -Sample parameter specifies that the highest-quality sampling should be used.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Scale: -Height=225, -Width=225, -Sample)]
[$MyImage->(Save: '/images/image.jpg')]
```

To contract an image:

Use the [Image->Scale] tag on a defined image variable. The following example contracts image.jpg to 25 x 25 pixels. The optional -Thumbnail parameter optimizes the image for the Web.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Scale: -Height=25, -Width=25, -Thumbnail)]
[$MyImage->(Save: '/images/image.jpg')]
```

To rotate an image:

Use the [Image->Rotate] tag on a defined image variable. The following example rotates the image 60 degrees clockwise on top of a white background.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Rotate: 60, -BGColor='FFFFFF')]
[$MyImage->(Save: '/images/image.jpg')]
```

To crop an image:

Use the [Image->Crop] tag on a defined image variable. The example below crops 10 pixels off of each side of a 70 x 70 image.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Crop: -Left=10, -Right=10, -Width=50, -Height=50)]
[$MyImage->(Save: '/images/image.jpg')]
```

To mirror an image:

Use the [Image->FlipV] tag on a defined image variable. The following example mirrors the image vertically.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->FlipV]
[$MyImage->(Save: '/images/image.jpg')]
```

Applying Image Effects

Lasso provides tags that allow you to add image effects by applying special image filters. This includes color modulation, image noise enhancement, sharpness controls, blur controls, contrast controls, and composite image merging. These tags are described below in *Table 7: Image Effects Tags*.

Table 7: Image Effects Tags

Tag	Description
[Image->Modulate]	Controls the brightness, saturation, and hue of an image. Brightness, saturation, and hue are controlled by three comma-delimited integer parameters, where 100 equals the original value.
[Image->Contrast]	Enhances the intensity differences between the lighter and darker elements of the image. Specify 'False' to reduce the image contrast, otherwise the contrast is increased.

<code>[Image->Blur]</code>	Applies either a motion or Gaussian blur to an image. To apply a motion blur, an <code>-Angle</code> parameter with a decimal degree value must be specified to indicate the direction of the motion. To apply a Gaussian blur, a <code>-Gaussian</code> keyword parameter must be specified in addition to <code>-Radius</code> and <code>-Sigma</code> parameters that require decimal values. The <code>-Radius</code> parameter is the radius of the Gaussian in pixels, and <code>-Sigma</code> is the standard deviation of the Gaussian in pixels. For reasonable results, the radius should be larger than the sigma.
<code>[Image->Sharpen]</code>	Sharpens an image. Requires <code>-Radius</code> and <code>-Sigma</code> parameters that require integer values. The <code>-Radius</code> parameter is the radius of the Gaussian sharp effect in pixels, and <code>-Sigma</code> is the standard deviation of the Gaussian sharp effect in pixels. For reasonable results, the radius should be larger than the sigma. Optional <code>-Amount</code> and <code>-Threshold</code> parameters may be used to add an unsharp masking effect. <code>-Amount</code> specifies the decimal percentage of the difference between the original and the blur image that is added back into the original, and <code>-Threshold</code> specifies the threshold in decimal pixels needed to apply the difference amount.
<code>[Image->Enhance]</code>	Applies a filter that improves the quality of a noisy, lower-quality image.

To adjust the brightness of an image:

Use the `[Image->Modulate]` tag on a defined image variable and adjust the first integer parameter, representing brightness. The following example increases the brightness of an image by a factor of two.

```
[Var: 'MyImage'=(Image: '/images/image.jpg')]
[$MyImage->(Modulate: 200, 100, 100)]
[$MyImage->(Save: '/images/image.jpg')]
```

To adjust the color saturation of an image:

Use the `[Image->Modulate]` tag on a defined image variable and adjust the second integer parameter, representing color saturation. The following example decreases the color saturation of an image by 25%.

```
[Var: 'MyImage'=(Image: '/images/image.jpg')]
[$MyImage->(Modulate: 100, 75, 100)]
[$MyImage->(Save: '/images/image.jpg')]
```

To adjust the hue of an image:

Use the [Image->Modulate] tag on a defined image variable and adjust the third integer parameter, representing hue. The following example tints the image green by increasing the hue value. Decreasing the hue value tints the image red.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Modulate: 100, 100, 175)]
[$MyImage->(Save: '/images/image.jpg')]
```

To adjust the contrast of an image:

Use the [Image->Contrast] tag on a defined image variable. The first example increases the contrast. The second example uses a False parameter, which reduces the contrast instead.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->Contrast]
[$MyImage->(Save: '/images/image.jpg')]

[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Contrast: 'False')]
[$MyImage->(Save: '/images/image.jpg')]
```

To apply a motion blur to an image:

Use the [Image->Blur] tag on a defined image variable. The following example applies a motion blur at 20 degrees.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Blur: -Angle=20)]
[$MyImage->(Save: '/images/image.jpg')]
```

To apply a Gaussian blur to an image:

Use the [Image->Blur] tag with the -Gaussian parameter on a defined image variable. The following example applies a Gaussian blur with a radius of 15 pixels and a standard deviation of 10 pixels.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Blur: -Radius=15, -Sigma=10, -Gaussian)]
[$MyImage->(Save: '/images/image.jpg')]
```

To sharpen an image:

Use the [Image->Sharpen] tag on a defined image variable. The following example applies a Gaussian sharp effect with a radius of 20 pixels and a standard deviation of 10 pixels.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Sharpen: -Radius=20, -Sigma=10)]
[$MyImage->(Save: '/images/image.jpg')]
```

To sharpen an image with an unsharp mask effect:

Use the [Image->Sharpen] tag with the -Amount and -Threshold parameters on a defined image variable. The following example applies an unsharp mask effect with a radius of 20 pixels and a standard deviation of 10 pixels.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Sharpen: -Radius=20, -Sigma=10, -Amount=50, -Threshold=20)]
[$MyImage->(Save: '/images/image.jpg')]
```

To enhance a low-quality image:

Use the [Image->Enhance] tag on a defined image variable.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->Enhance]
[$MyImage->(Save: '/images/image.jpg')]
```

Adding Text to Images

Lasso allows text to be overlaid on top of images using the [Image->Annotate] tag, as described below in the following table.

Table 8: Annotate Image Tag

Tag	Description
[Image->Annotate]	Overlays text on to an image. Requires a string value as a parameter, which is the text to be overlaid. Required -Left and -Top parameters specify the place of the text in pixel integers relative to the upper left corner of the image. An optional -Font parameter specifies the name (with extension) and full path to a system font to be used for the text, and an optional -Size parameter specifies the text size in integer pixels. An optional -Color parameter specifies the text color as a hex string ('#FFCCDD'). An optional -Aliased keyword parameter turns on text anti-aliasing.

Fonts Note: When specifying a font, the full hard drive path to the font must be used (e.g. -Font='/Library/Fonts/Arial.ttf'). True Type (.ttf), and Type One (.pfa, .pfb) font types are officially supported.

To add text to an image:

Use the [Image->Annotate] tag on a defined image variable. The example below adds the text (c) 2003 Blue World Communications to the specified image.

```
[Var: 'MyImage' =(Image: '/images/image.jpg')]
[$MyImage->(Annotate: '(c) 2003 Blue World Communications', -Left=5, -Top=300,
               -Font='/Library/Fonts/Arial.ttf', -Size=8, -Color='#000000', -Aliased)]
[$MyImage->(Save: '/images/image.jpg')]
```

Merging Images

Lasso allows images to be merged using the [Image->Composite] tag. The [Image->Composite] tag supports over 20 different composite methods, which are described in the following tables.

Table 9: Composite Image Tag

Tag	Description
[Image->Composite]	Composites a second image onto the current image. Requires a second LDML image variable to be composited. An -Op parameter specifies the composite method which affects how the second image is applied to the first image (a list of operators is shown below). Optional -Left and -Top parameter specify the horizontal and vertical offset of the second image over the first in integer pixels (defaults to the upper left corner). An optional -Opacity parameter attenuates the opacity of the composited second image, where a value of 0 is fully opaque and 1.0 is fully transparent.

The table below shows the various composite methods that can be specified in the -Op parameter of the [Image->Composite] tag. The descriptions for each method are adapted from the ImageMagick Web site.

Table 10: Composite Image Tag Operators

Composite Operator	Description
Over	The result is the union of the the two image shapes with the composite image obscuring the image in the region of overlap.
In	The result is the first image cut by the shape of the second image. None of the second image data is included in the result.
Out	The result is the second image cut by the shape of the first image. None of the first image data is included in the result.

Plus	The result is the sum of the raw image data with output image color channels cropped to 255.
Minus	The result is the subtraction of the raw image data with color channel underflow cropped to zero.
Add	The result is the sum of the raw image data with color channel overflow channel wrapping around 256.
Subtract	The result is the subtraction of the raw image data with color channel underflow wrapping around 256.
Difference	Returns the difference between two images. This is useful for comparing two very similar images.
Bumpmap	The resulting image is shaded by the second image.
CopyRed	The resulting image is the red layer in the image replaced with the red layer in the second image.
CopyGreen	The resulting image is the green layer in the image replaced with the green layer in the second image.
CopyBlue	The resulting image is the blue layer in the image replaced with the blue layer in the second image.
CopyOpacity	The resulting image is the opaque layer in the image replaced with the opaque layer in the second image.
Displace	Displaces part of the first image where the second image is overlaid.
Threshold	Only colors in the second image that are darker than the colors in the first image are overlaid.
Darken	Only dark colors in the second image are overlaid.
Lighten	Only light colors in the second image are overlaid.
Colorize	Only base spectrum colors in the second image are overlaid.
Hue	Only the hue of the second image is overlaid.
Saturate	Only the saturation of the second image is overlaid.
Luminize	Only the luminosity of the the second image is overlaid.
Modulate	Has the effect of Hue, Saturate, and Luminize functions applied at the same time.

To overlay an image on top of another image:

Use the [Image->Composite] tag to add a defined image variable to a second defined image variable. The following example adds image2.jpg offset by five pixels in the upper left corner of image1.jpg.

```
[Var: 'MyImage1' =(Image: '/images/image1.jpg')]
[Var: 'MyImage2' =(Image: '/images/image2.jpg')]
[$MyImage1->(Composite: $MyImage2, -Left=5, -Top=5)]
[$MyImage1->(Save: '/images/image1.jpg')]
```

To add a watermark to an image:

Use the [Image->Composite] tag with the -Opacity parameter to add a defined image variable to a second defined image variable. The following example adds a mostly transparent version of image2.jpg to image1.jpg.

```
[Var: 'MyImage1'=(Image: '/images/image1.jpg')]
[Var: 'MyImage2'=(Image: '/images/image2.jpg')]
[$MyImage1->(Composite: $MyImage2, -Opacity=0.75)]
[$MyImage1->(Save: '/images/image1.jpg')]
```

To shade image with a second image:

Use the [Image->Composite] tag with the Bumpmap operator to shade a defined image variable over a second defined image variable.

```
[Var: 'MyImage1'=(Image: '/images/image1.jpg')]
[Var: 'MyImage2'=(Image: '/images/image2.jpg')]
[$MyImage1->(Composite: $MyImage2, -Op='Bumpmap')]
[$MyImage1->(Save: '/images/image1.jpg')]
```

To return the pixel difference between two images:

Use the [Image->Composite] tag with the Difference operator to return the pixel difference between two defined image variables.

```
[Var: 'MyImage1'=(Image: '/images/image1.jpg')]
[Var: 'MyImage2'=(Image: '/images/image2.jpg')]
[$MyImage1->(Composite: $MyImage2, -Op='Difference')]
[$MyImage1->(Save: '/images/image1.jpg')]
```

Extended ImageMagick Commands

For users who have experience using the ImageMagick command line utility, Lasso provides the [Image->Execute] tag to allow advanced users to take advantage of additional ImageMagick commands and functionality.

Table 11: ImageMagick Execute Tag

Tag	Description
[Image->Execute]	Execute ImageMagick commands. Provides direct access to the ImageMagick command-line interface. Supports the Composite, Mogrify, and Montage commands.

For detailed descriptions of the Composite, Mogrify, and Montage commands and corresponding parameters, see the following URL.

<http://www.imagemagick.com/www/utilities.html>

To execute an ImageMagick command using Lasso:

Use the [Image->Execute] tag on a defined image variable, with the desired command as the parameter. The following example shows the Mogrify command for resizing an image.

```
[Var: 'MyImage' =(Image: '/images/image.gif')]
[$MyImage->(Execute: 'mogrify -resize 640X480! image.gif')]
[$MyImage->(Save: '/images/image.gif')]
```

Serving Image and Multimedia Files

This section discusses how to serve image and multimedia files, including referencing files within HTML pages and serving files separately via HTTP.

Referencing Within HTML Files

The easiest way to serve images and multimedia files is simply by referencing files stored within the Web server root using standard HTML tags such as or <embed>. The path to the image file can be calculated in the format file or stored within a database field. Since the specified file is ultimately served by the Web server application which is optimized for serving images and multimedia files, this is the most efficient way to serve images and multimedia files.

To generate the path to an image or multimedia file:

- The following example shows a variable Company_Name that contains blueworld. This variable is used to construct a path to an image file stored within the Images folder named with the company name and _logo.gif to form the full file path /Images/blueworld_logo.gif.

```
[Variable: 'Company_Name'='blueworld']

```

→

- The following example shows a variable Company_Name that contains blueworld. This variable is used to construct a path to an image file stored within the Images folder named with the company name and _logo.gif to form the full file path /Images/blueworld_logo.gif. The path to the image file

is stored within the variable `Image_Path` and then reference in the HTML `` tag.

```
[Variable: 'Company_Name'='blueworld']
[Variable: 'Image_Path'='/Images/' + $Company_Name + '_logo.gif']

```

→ ``

- The following example shows a variable `Band_Name` that contains `ArtOfNoise`. This variable is used to construct a path to sound files stored within the `Sounds` folder named with the band name and `.mp3` to form the full file path `/Sounds/ArtOfNoise.mp3`. The path to the sound file is stored within the variable `Sound_Path` and then reference in the HTML `<a>` link tag.

```
[Variable: 'Band_Name'='ArtOfNoise']
[Variable: 'Sound_Path'='/Images/' + $Band_Name + '.mp3']
<a href="[Variable: 'Sound_Path']">Download MP3</a>
```

→ `Art of Noise Song`

Serving Files via HTTP

Lasso can also be used to serve image and multimedia files rather than merely referencing them by path. Files are served through Lasso using the `[File_Serve]` tag or a combination of the `[Content_Type]` tag and `[Include_Raw]` tags. LDML 7 also includes an `[Image->Data]` tag that automatically converts an image variable to a binary string, allowing an edited `[Image]` variable to be output by `[File_Serve]` without it first being written to file.

In order to serve an image or multimedia file through Lasso the MIME type of the file must be determined. Often, this can be discovered by looking at the configuration of the Web server or Web browser. The MIME type for a GIF is `image/gif` and the MIME type for a JPEG is `image/jpeg`.

Note: It is not recommended that you configure your Web server application to process all `.gif` and `.jpg` files through Lasso. Lasso will attempt to interpret the binary data of the image file as LDML code. Instead, use one of the procedures below to serve an image file with a `.lasso` extension.

Table 12: Image Serving Tag

Tag	Description
[File_Serve]	Serves a file in place of the output of the current format file. The first parameter is the data to be served. Optional -File parameter specifies the name of the served data. Optional -Type parameter allows the MIME type to be overridden from the default of text/html.
[Image->Data]	Converts an image variable to a binary string. This is useful for serving images to a browser without writing the image to file.

To serve an image file:

- Use the [File_Serve] tag to set the MIME type of the image to be served, and use the [Image->Data] tag to get the binary data from a defined [Image] variable. The [File_Serve] tag aborts the current file so it must be the last tag to be processed. The following example shows a GIF named Picture.gif being served from an Images folder.

```
[Var:'Image'=(Image: '/Images/Picture.gif')]
[File_Serve: $Image->Data, -Type='image/gif']
```

- Use the [Content_Type] tag to set the MIME type of the image to be served and use the [Include_Raw] tag to include data from the image file. The two tags should be the only content of the file and should not be separated by any white space. The following example shows a GIF named Picture.gif being served from an Images folder.

```
[Content_Type: 'image/gif'][Include_Raw: '/Images/Picture.gif'][Abort]
```

If either of the code examples above is stored in a file named Image.lasso at the root of the Web serving folder then the image could be accessed with the following tag.

```

```

To serve a multimedia file:

Use the [Content_Type] tag to set the MIME type of the file to be served and use the [Include_Raw] tag to include data from the multimedia file. The two tags should be the only content of the file and should not be separated by any white space. The following example shows a sound file named ArtOfNoise.mp3 being served from a Sounds folder.

```
[Content_Type: 'audio/mp3'][Include_Raw: '/Sounds/ArtOfNoise.mp3'][Abort]
```

If the code above is stored in a file named ArtOfNoise.lasso at the root of the Web serving folder then the sound file could be accessed with the following <a> link tag.

```
<a href="http://www.example.com/ArtOfNoise.lasso">Art of Noise Song</a>
```

This same technique can be used to serve multimedia files of any type by designating the appropriate MIME type in the [Content_Type] tag.

To serve an image file with a proper file extension:

The following example demonstrates how to serve a GIF file with a .gif extension. The extension .gif must be allowed in Lasso Administration. Use the [Content_Type] tag to set the MIME type of the image to be served and use the [Include_Raw] tag to include data from the image file.

```
[Content_Type: 'image/gif'][Include_Raw: '/Images/Picture.gif'][Abort]
```

The file will need to be referenced using Action.Lasso and a -Response command tag within the URL. If the code above is stored in a file named Image.gif at the root of the Web serving folder then the image could be accessed with the following tag.

```

```

To serve a multimedia file with a proper file extension:

The following example demonstrates how to serve a sound file with a .mp3 extension. The extension .mp3 must be allowed in Lasso Administration. Use the [Content_Type] tag to set the MIME type of the multimedia file to be served and use the [Include_Raw] tag to include data from the multimedia file.

```
[Content_Type: 'audio/mp3'][Include_Raw: '/Sounds/ArtOfNoise.mp3'][Abort]
```

The file will need to be referenced using Action.Lasso and a -Response command tag within the URL. If the code above is stored in a file named ArtOfNoise.mp3 at the root of the Web serving folder then the image could be accessed with the following <a> link tag.

```
<a href="http://www.example.com/Action.Lasso?-Response=ArtOfNoise.mp3">
  Art Of Noise Song
</a>
```

This same technique can be used to serve multimedia files of any type by designating the appropriate MIME type in the [Content_Type] tag.

To limit access to a file:

Since the format file can process any LDML code before serving the image it is easy to create a file that generates an error if an unauthorized person tries to access a file. The following code checks the [Client_Username] for the name John. If the current user is not named John then a file Error.gif is served instead of the desired Picture.gif file.

```
<?LassoScript
  Content_Type: 'image/gif';
  If: (Client_Username) == 'John';
    Include_Raw: '/Images/Picture.gif';
  Else;
    Include_Raw: '/Images/Error.gif';
  /If;
?>
```

This same technique can be used to restrict access to any image or multimedia file. It could actually be used to restrict access to any format file.

27

Chapter 27

HTTP/HTML Content and Controls

This chapter describes the tags which can be used to send and receive files with remote HTTP and FTP servers, include files from remote HTTP and HTTPS servers, interpret HTTP requests, alter the HTTP headers of responses, and to redirect clients to another URL.

- *Include URLs* describes how to include files from remote Web servers, including SSL-protected servers.
- *Redirect URL* describes how to forward clients to a different URL..
- *HTTP Tags* describes how to perform HTTP requests to another Web server.
- *FTP Tags* describes how to perform FTP requests to an FTP server.
- *Cookie Tags* describes how to set and retrieve cookies from a Web client
- *Caching Tags* describes how to cache format file content using Lasso.
- *Server Push* describes how to enable progressive download of HTML pages.
- *Header Tags* describes the tags which allow the current HTTP response headers to be modified.
- *Request Tags* describes the tags which return information about the current HTTP request and allow the HTTP header of the response to be manipulated.
- *Client Tags* describes the tags which return information about the current Web client.
- *Server Tags* describes the tags which return information about the current Web server.

Include URLs

The [Include_URL] tag allows data from another Web server to be included into the contents of a page which is being served to a visitor. This can include HTTP or HTTPS servers. The [Include_URL] tag is replaced by the contents of the remote Web page. Optional parameters allow GET or POST parameters, authentication information, or extra MIME headers to be sent along with the request. Other optional parameters also allow for the MIME headers of the response to be retrieved.

The [Include_URL] tag can be used for any of these purposes.

- To fetch a remote Web page to show to a site visitor. Lasso can be used as a proxy that retrieves the remote page, performs some processing, then sends the page to the visitor.
- To incorporate a portion of a remote Web page into a Lasso format file. A remote Web page can be retrieved, the desired content extracted, and placed into a Lasso format file.
- [Include_URL] can also be used in pages on the same Web server in which Lasso is running.
- To trigger an action in a remote Web application server. [Include_URL] could be used to trigger a CGI on another Web server.
- To trigger an action in a remote Web application server protected via SSL. [Include_URL] could be used to initiate a credit card transaction at a secure HTTPS processing site.
- To trigger an action on the same Web server in which Lasso is running. The [Event_Schedule] tag uses [Include_URL] to call format files at the designated time.

Implementation Note: The [Include_URL] tag is implemented in LDML 7 using libCURL 7.9.5 with OpenSSL for communication with HTTP and HTTPS servers. For more information on libCURL, visit <http://curl.sourceforge.net>. For more information on OpenSSL, visit <http://www.openssl.org>.

Table 1: Include URL Tag

Tag	Description
[Include_URL]	Includes a Web page from a remote HTTP or HTTPS server, or from the local server. Requires the target URL as a value parameter and accepts many optional parameters.

The [Include_URL] tag accepts many parameters which define how the remote page should be fetched. These are summarized in *Table 2: [Include_URL] Parameters*.

Code which is returned by [Include_URL] will be HTML encoded by default. Specify -EncodeNone so fetched HTML code will be rendered as part of your Web page. Code which is included with [Include_URL] will not undergo further processing by Lasso unless the [Process] tag is called explicitly on the results.

Table 2: [Include_URL] Parameters

Parameter	Description
URL	Specifies the URL which is to be fetched. Required.
-POSTParams	Specifies an array or map of POST parameters. Optional. The request will be sent using the POST method if this parameter is included. -POSTParams can also be used to post a string to a remote server.
-GETParams	Specifies an array or map of GET parameters. Optional.
-SendMIMEHeaders	Specifies an array of additional MIME headers that should be included with the request. Optional.
-Username	Specifies the username that should be used to authenticate the request. Optional.
-Password	Specifies the password that should be used to authenticate the request. Passwords are encoded in Base 64. Optional.
-RetrieveMIMEHeaders	Specifies the name of a variable which will be set to an array containing all of the MIME headers in the response. Optional.
-NoData	Specifies that the data from the request should not be returned. Optional.

To include a URL into the current format file:

Use the [Include_URL] tag with the URL of the remote page. The following example shows how to include Blue World's front page.

```
[Include_URL: 'http://www.blueworld.com/']
```

The port of a Web server may also be specified in the URL.

```
[Include_URL: 'http://www.blueworld.com:80/']
```

```
[Include_URL: 'http://www.blueworld.com:1024/']
```

```
[Include_URL: 'http://www.blueworld.com:8180/']
```

To include a URL from a password-protected HTTPS server into the current format file:

Use the [Include_URL] tag with the -Username and -Password parameters (recommended). The following example shows how to include an SSL-protected page.

```
[Include_URL: 'https://store.example.com/', -Username='my_username',
-Password='my_password']
```

This can also be achieved without the use of the -Username and -Password parameters by submitting the username and password in the URL.

```
[Include_URL: 'my_username:my_password@https://store.example.com']
```

To simulate an HTML form submission:

An HTML form submission can be simulated using the [Include_URL] tag with the -POSTParams parameter. The inputs of the form should be included as name/value parameters within an array. The following form is shown below as an equivalent [Include_URL] tag.

```
<form action="http://www.example.com/response.lasso" method="POST">
  <input type="hidden" name="-Database" value="Example">
  <input type="hidden" name="-Table" value="Contacts">
  <input type="hidden" name="-KeyField" value="ID">
  <p><input type="submit" name="-FindAll" value="Find All Records"></p>
</form>
```

The form inputs are assembled into an array as follows.

```
[Variable: 'POST_Params' = (Array: -Database='Example',
-Table='Contacts', -KeyField='ID', -FindAll='Find All Records')]
```

This variable can then be included in the [Include_URL] tag. The following tag will include the contents of the format file response.lasso with the results of the -FindAll action.

```
[Include_URL: 'http://www.example.com/response.lasso',
-POSTParams=(Variable: 'POST_Params')]
```

To process an included URL:

Often it is necessary to do some post-processing on an included URL in order to extract a portion of the page. Most included pages will have <html>, <head> and <body> tags which are redundant since they are specified in the format file which is including the remote page. The following code extracts everything between the opening <body> tag and closing </body> tag using a regular expression.

```
[Variable: 'Page_Text' = (Include_URL: 'http://www.blueworld.com/')]
[String_ReplaceRegExp: (Variable: 'Page_Text'),
-Find='.*<body[^>]*>(.*?)</body>.*',
-Replace='\1']
```

The regular expression will match the entire included file. `.*` will match all characters until the `<body>` tag which is written as `<body[^>]*>` so that any parameters of the `<body>` tag will be included. The parenthesized expressions `(.*?)` matches the contents of the body tag which is ended by `</body>`. Finally, `.*` matches all other characters until the end of the file.

The replacement is simply `\1` which replaces the entire expression with the contents of the first parenthesized expression, the contents of the `<body> ... </body>` tags.

Redirect URL

The `[Redirect_URL]` tag can be used to send a client to a different URL. Processing of the current page stops as soon as `[Redirect_URL]` is called (except for `[Handle] ... [/Handle]` tags which execute normally). `[Redirect_URL]` works by altering the HTTP response header which is returned to the client. The use of `[Redirect_URL]` may override specific settings made using the `[Header] ... [/Header]` tags. `[Redirect_URL]` cannot be used on a page below a `[Server_Push]` tag.

The parameter to `[Redirect_URL]` must be a full URL and should include the explicit protocol, e.g. `http://`. Specifying an absolute or relative path to another format file on the Web server will not work. For example, to reference the home page of the Web server `www.example.com`, the following full URL would be used.

```
http://www.example.com/default.lasso
```

Table 3: Redirect URL Tag

Tag	Description
<code>[Redirect_URL]</code>	Accepts a single parameter which is a URL to which the client should be sent.

To redirect a client to another page:

Specify the full URL of the page to which the client should be redirected within a `[Redirect_URL]` tag.

- The following examples show how to redirect a client to Microsoft's or Apple's Web sites.

```
[Redirect_URL: 'http://www.microsoft.com/']
```

```
[Redirect_URL: 'http://www.apple.com/']
```

- The following example shows how to redirect a client to the login page login.lasso contained in the root folder of the www.example.com Web site.

```
[Redirect_URL: 'http://www.example.com/login.lasso']
```

- The following example shows how to redirect a client to another page redirect.lasso in the same folder as the current page. The [Server_Name] and [Response_Path] tags are used to return the name of the current server and the path to the current response page.

```
[Redirect_URL: 'http://' + (Server_Name) + (Response_Path) + 'redirect.lasso']
```

HTTP Tags

LDML 7 provides HTTP protocol tags that allow developers to send and receive files via HTTP. These tags can generally be used for programmatically uploading and downloading files to and from another Web server.

Implementation Note: The [HTTP_...] tags are implemented in LDML 7 using libCURL 7.9.5. For more information on libCURL, visit <http://curl.sourceforge.net>.

Table 4: HTTP Tags

Tag	Description
[HTTP_GetFile]	Downloads a file from a remote HTTP server. Requires the -URL parameter, which is the URL from which the file will be downloaded, and the -File parameter, which is the name and path to the local file to be created. Optional -Username and -Password parameters may be used to specify a username and password needed to log in to the remote HTTP server. This tag is similar to [Include_URL], except that the file is written to disk rather than being output inside a Lasso format file.

File Permissions Note: The current Lasso user must have adequate file and folder permissions to copy a file or write to folder on the local machine. The same file permissions are required for [HTTP_...] tags as the [File_...] tags. See *Chapter 20: Files and Logging* for more information.

To download a file from an HTTP server:

Use the [HTTP_GetFile] tag. The following example downloads a file named download.zip from the URL <http://www.example.com/download.zip> to the local documents folder.

```
[HTTP_GetFile: -URL='http://www.example.com/download.zip', -File='/documents/
download.zip']
```

To download a file from a password-protected HTTP server on a non-default port:

Use the [HTTP_GetFile] tag with the -Username and -Password parameters. The following example downloads a file named download.zip at <http://www.example.com:1024/private/download.zip> where a username and password are required to access to the private folder.

```
[HTTP_GetFile: -URL='http://www.example.com:1024/private/download.zip', -File='/
documents/download.zip', -Username='my_username', -Password='my_password']
```

FTP Tags

LDML 7 also provides FTP protocol tags that allow developers to send and receive files via FTP. These tags can generally be used for programmatically uploading and downloading files to and from an FTP server.

Note: These tags do not make Lasso Professional 7 an FTP server, but allow Lasso Professional 7 to put and get files from other FTP servers similar to an FTP client.

Implementation Note: The [FTP_...] tags are implemented in LDML 7 using libCURL 7.9.5. For more information on libCURL, visit <http://curl.sourceforge.net>.

Table 5: FTP Tags

Tag	Description
[FTP_PutFile]	Uploads a local file up to a remote FTP server. Requires the -URL parameter, which is the URL folder and file name of the file to be uploaded, and the -File parameter, which is the path to the local file to be uploaded. Optional -Username and -Password parameters may be used to specify a username and password needed to log in to the remote FTP server.

[FTP_GetFile]	Downloads a file from a remote FTP server. Requires the -URL parameter, which is the URL from which the file will be downloaded, and the -File parameter, which is the name and path to the local file to be created. Optional -Username and -Password parameters may be used to specify a username and password needed to log in to the remote FTP server.
[FTP_GetListing]	Lists all files accessible to the current user in the remote FTP server URL folder. Outputs an array of maps for each file entry containing the file name, type (directory, file, or link), modification date/time, and size in bytes (for files only). Requires the -URL parameter, which is the URL of the folder to be listed. Optional -Username and -Password parameters may be used to specify a username and password needed to log in to the remote FTP server. The username and password values often determine which files are shown by the FTP server.

File Permissions Note: The current Lasso user must have adequate file and folder permissions to copy a file or write to a folder on the local machine. The same file permissions are required for [FTP_...] tags as the [File_...] tags. See *Chapter 20: Files and Logging* for more information.

To upload a file to an FTP server:

Use the [FTP_PutFile] tag. The following example uploads a file named myfile.zip to the URL ftp://ftp.example.com.

```
[FTP_PutFile: -URL='ftp://ftp.example.com/myfile.zip', -File='/documents/myfile.zip']
```

To upload a file to a password-protected FTP server:

Use the [FTP_PutFile] tag with the -Username and -Password parameters. The following example uploads a file named myfile.zip to ftp://ftp.example.com/private/ which requires a username and password to access the private folder.

```
[FTP_PutFile: -URL='ftp://ftp.example.com/private/myfile.zip', -File='/documents/myfile.zip', -Username='my_username', -Password='my_password']
```

To download a file from an FTP server:

Use the [FTP_GetFile] tag. The following example downloads a file named download.zip from the URL ftp://ftp.example.com/download.zip.

```
[FTP_GetFile: -URL='ftp://ftp.example.com/download.zip', -File='/documents/download.zip']
```

To download a file from a password-protected FTP server:

Use the [FTP_GetFile] tag with the -Username and -Password parameters. The following example downloads a file named download.zip from ftp://ftp.example.com/private/download.zip where a username and password are required to access the private folder.

```
[FTP_GetFile: -URL='ftp://ftp.example.com/private/download.zip', -File='/documents/
download.zip', -Username='my_username', -Password='my_password']
```

To list all files available in a folder on a password-protected FTP server:

Use the [FTP_GetListing] tag with the -Username and -Password parameters. The following example lists all files in the ftp://ftp.example.com/private/ folder that are available to the my_username user.

```
[FTP_GetListing: -URL='ftp://ftp.example.com/private/', -Username='my_username',
-Password='my_password']
```

```
→ [Array: (Map: 'FileName'='download.zip',
               'FileSize'='101k',
               'FileType'='File',
               'FileType'='2002-09-29 15:30:00'),
  (Map: 'FileName'='More_Files',
        'FileType'='File',
        'FileType'='2002-09-12 12:14:39')]
```

Note: The modification date for each file using the [FTP_GetListing] tag will be returned using the date format that is used by the remote FTP server.

Cookie Tags

Cookies allow small amounts of information to be stored in the Web browser by Lasso. Each time another page on the same server is loaded, all stored cookies are sent back to Lasso. Multiple cookies can be stored in a client's Web browser and then retrieved on subsequent pages. Cookies can be used to store a client's authentication information, customer ID, site preferences, or even an entire shopping cart. Lasso's sessions make automatic use of cookies to store each client's session ID so server-side variables can be made persistent from page to page.

Please see *Chapter 19: Sessions* for an introduction to sessions and the *Cookies* section of *Chapter 2: Web Application Fundamentals* for a technical introduction to cookies.

Cookies are reliant on support from the client's Web browser for much of their functionality. Preferences for when cookies expire and what domains

can retrieve a cookie can be established using the [Cookie_Set] tag, but those preferences must be enforced by the client's Web browser in order for them to have any effect. Clients can even turn off cookie support altogether in most Web browsers.

Cookies are communicated to and from the Web server in the HTTP response header and subsequent HTTP requests. Cookies are not available in the page within which they were set, they are only available in subsequent pages loaded by the same client.

Table 6: Cookie Tags

Tag	Description
[Cookie]	Returns the value for a named cookie. Accepts one required parameter, the name of the cookie whose value should be returned.
[Cookie_Set]	Sets a cookie with a given name and value. See the table below for details about this tag's parameters.
[Client_CookieList]	Returns a string which contains every cookie sent along with the current HTTP request.
[Client_Cookies]	Returns a pair array containing every cookie sent along with the current HTTP request.

Setting Cookies

Cookies are set using the [Cookie_Set] tag. The one required parameter of the tag is a user-defined name/value parameter specifying the name of the cookie and the value which is to be stored under that name. However, it is recommended that all parameters of the [Cookie_Set] tag be specified in order to ensure compatibility with the greatest range of Web browsers.

Table 7: [Cookie_Set] Parameters

Tag	Description
Name/Value	A name/value parameter defines the name of the cookie and the value which should be stored under that name. Required.
-Expires	The number of minutes until the cookie expires. Optional. If left blank, most cookies expire when the client quits their Web browser application. A negative value instructs a Web browser to expire a cookie immediately.
-Domain	The domain of the cookie. Cookies will only be sent to servers with this domain. Optional, but recommended.
-Path	The path of the cookies. Cookies will only be sent to pages which are in subfolders of this path. Optional, but recommended.
-Secure	If specified then the cookie will only be transmitted back through secure HTTPS protocol.

The total number of characters of the name/value parameter and all other parameters of the [Cookie_Set] tag must be less than 2048 characters. The name of the cookie must be less than 1024 characters. The value of the cookie must be less than 1024 characters. The -Expires parameter should be no more than 10 digits. The -Path and -Domain parameters should be no more than 256 characters each.

Note: The parameters required for the [Cookie_Set] tag vary depending on what Web clients are being used by site visitors. In general, it is safest to specify each of -Expires, -Domain, and -Path in order to ensure maximum compatibility.

To set a cookie:

Use the [Cookie_Set] tag with each of the parameters defined. The following example shows how to create a cookie named Cookie_Name with the value Cookie_Value for the domain example.com with an expiration time of 24 hours (1440 minutes). The path is set to / so all pages in the site will have access to the cookie.

```
[Cookie_Set: 'Cookie_Name'='Cookie_Value'
-Domain='example.com',
-Path='/',
-Expires=1440]
```

The example above shows -Domain set to example.com. Setting -Domain to the name of the domain rather than to the name of a particular Web server ensures that any server within the domain can retrieve the cookie.

For example, mail.example.com and images.example.com could retrieve Cookie_Name set above.

To set a cookie that can be retrieved by another Web server:

The -Domain parameter can be used to define that a cookie be returned to another Web server. This can be useful for interactions where a customer needs to be tracked as they move between different Web servers.

The following example shows how a cookie can be set so that it will be served to the server www.otherserver.com. This cookie will not be sent to subsequent pages loaded on the current server by the client. It will only be served to www.otherserver.com when they visit that Web server.

```
[Cookie_Set: 'Cookie_Name'='Cookie_Value'
-Domain='otherserver.com',
-Path='/',
-Expires=1440]
```

Note: Many Web browsers have a preference which prohibits cookies being set which will be read by a different server. If the -Domain parameter is not specified to the same domain as the Web server hosting Lasso Service then the cookie may not be set in all Web browsers.

To delete a cookie:

Cookies can be deleted by setting the -Expires parameter to a negative number or by resetting the value of the cookie. It is good practice to delete cookies which are no longer needed. Some Web browsers do not delete expired cookies properly so extra data may end up being sent to the Web server with every URL request. The following example shows how to delete the cookie Cookie_Name by setting it to the empty string "" and setting its expiration to -1.

```
[Cookie_Set: 'Cookie_Name'="",
-Domain='example.com',
-Path='/',
-Expires=-1]
```

Retrieving Cookies

Cookies are retrieved by name. However, only cookies which were sent by the client's Web browser along with the current HTTP request can be retrieved by Lasso. The Web browser determines what cookies to send based on the domain, path, and expiration set for each cookie. The implementation differs from browser to browser so some client's may not support all types of cookies.

To retrieve a cookie:

If a cookie is available it can be retrieved using the [Cookie] tag. This tag accepts a single parameter which is the name of the cookie to be retrieved. The tag will return an empty string if the cookie is not defined. The following code returns the value Cookie_Value for the cookie Cookie_Name.

```
[Cookie: 'Cookie_Name'] → Cookie_Value
```

To retrieve all cookies:

There are two ways to retrieve a list of all cookies that have been sent along with the current HTTP request.

- Use [Client_Cookies] to return an array of all cookies set for the current HTTP request. [Client_Cookies] returns a pair array where each pair contains the name and value of a cookie. The following example shows how to display all cookies that are currently set using [Loop] ... [/Loop] tags. The result is the single Cookie_Name with value Cookie_Value.

```
[Loop: (Client_Cookies->Size)]
  [Variable: 'Temp_Cookie' = (Client_Cookies->(Get: (Loop_Count)))]
  <br>[Output: $Temp_Cookie->First + ': ' + $Temp_Cookie->Second]
[/Loop]
```

→
Cookie_Name: Cookie_Value

- Use [Client_CookieList] to return a string that contains the names and values of all cookies set for the current HTTP request. This tag can be used for debugging purposes to quickly display a list of all cookies. The cookies are returned separated by semi-colons ; with the name of the cookie separated from the value by an equal sign =. The following example shows a single cookie Cookie_Name with value Cookie_Value.

```
[Client_CookieList] → Cookie_Name: Cookie_Value;
```

To check if a cookie is set:

Use the [Array->Find] tag to search through the [Client_Cookies] tag. If the [Array->Find] returns a pair value then the cookie is set. If it returns Null then the cookie is not set. Using this method is more reliable than simply calling [Cookie] with the name of a cookie since it is impossible to tell whether a returned value of the empty string "" is due to a cookie not being set or due to a cookie being set to the empty string. The following example returns True since the cookie Cookie_Name is set.

```
[If: (Client_Cookies->(Find: 'Cookie_Name')) != Null] True [/If]
```

→ True

Checking for Cookie Support

Since cookies can be deactivated within a client's Web browser it is important to check whether cookies are supported before allowing a client to view portions of a Web site that require cookies. The following code will perform a check for cookie support by setting a cookie and then redirecting the client to another page which checks the cookie value.

To check whether cookies are supported:

The page `cookie_set.lasso` contains a `[Cookie_Set]` tag that sets a cookie `Test_Cookie` to the value `Test_Value` and a `[Redirect_URL]` tag which sends the client to the page `cookie_check.lasso`.

```
[Cookie_Set: 'Test_Cookie'='Test_Value',
  -Domain='example.com',
  -Path='/']
[Redirect_URL: 'http://www.example.com/cookie_check.lasso']
```

The page `cookie_check.lasso` checks to see if the cookie is set using the `[Array->Find]` tag on `[Client_Cookies]`. If it is set, it redirects the user to the default page of the Web site `default.lasso`. If cookies are not supported then the user is redirected to the page `error.lasso` which contains a warning message.

```
[If: (Client_Cookies)->(Find: 'Cookie_Name') != Null]
  [Redirect_URL: 'http://www.example.com/default.lasso']
[Else]
  [Redirect_URL: 'http://www.example.com/error.lasso']
[/If]
```

Caching Tags

New content caching tags in Lasso Professional 7 allow a portion of a page to be cached either to a global variable or to a session. Lasso is able to cache the output of dynamic LDML code and data source queries, as well as the values of named LDML variables for later use. These tags allow developers to reduce database and server load by having Lasso only recalculate various portions of a page periodically.

The first time a page with `[Cache]` ... `[/Cache]` tags is hit, the contents of the tags are remembered for a specified period of time. The Lasso cache can be set to refresh itself at scheduled time intervals, or when certain conditions are met.

Important: When using the cache tags, it is important to know that any dynamic changes that occur in cached LDML code will be ignored, and only the original cached values will be output until the cache expires.

Caching Output Values

Lasso allows the values output by dynamic LDML code to be cached so that the dynamic operations (such as a data query) are not performed again until the cache expires or is dumped. This is accomplished by surrounding the code to cache with the [Cache] ... [/Cache] container tags, which are described below.

Table 8: [Cache] Tag

Tag	Description
[Cache] ... [/Cache]	Container tag used for caching elements on a page in Lasso's internal cache. Requires a -Name parameter which specifies the name of the cache, and several optional parameters may be used as shown in the following table.

The optional parameters for the [Cache] ... [/Cache] container tags are described in *Table 9: [Cache] Tag Parameters*.

Table 9: [Cache] Tag Parameters

Tag	Description
-Name	Specifies the name of the cache. The name of the cache identifies the cached contents so it can be referenced on several pages. This is the only required parameter.
-Expires	Specifies how many seconds the cached contents should last. If the cached contents is older than the time interval specified, then new content values will be cached on the next page load.
-Condition	Allows arbitrary refresh conditions to be specified. Accepts a boolean value of True or False and refreshes the contents immediately when True. Conditional expressions may be used to output the required True or False value. For example, -Condition=((Action_Param: 'Refresh') == 'Yes') refreshes the cache if the action param is equal to Yes.
-Session	Specifies the name of a session that the cached content should be stored in. This allows the cached content to be user specific. If no -Session parameter is specified, then the content will be stored in a global variable instead.

-UseGlobal	Can be used in concert with -Session to store cached data in both a global variable and a session. All caches are stored in a global variable by default if neither -Session or -UseGlobal is specified.
-Key	Can be used to secure a cache on a server. Requires a password string value. Once a cache has been created with a -Key parameter and value, the cache will only be returned if subsequent [Cache_...] tags contain the same -Key parameter and value. Optional.

Restart Note: Caches stored in global variables do not persist between restarts. They must be refreshed the first time they are hit on a Lasso page. Global variables are used by default unless a -Session parameter is specified.

To cache content with an expiration:

Surround the portion of your page that you wish to cache with the [Cache] ... [/Cache] container tags using the -Expires parameter. In the example below, the output of the data source query surrounded by the [Cache] ... [/Cache] tags will be stored in a global variable, and then output consistently with each page refresh (without performing the data source query again) until the cache expires 3600 seconds later.

```
[Cache:
  -Name='Cache_Name',
  -Expires=3600]

[Inline: -Database='Contacts', -SQL='Select * from people where ID < 3']
  [Field:'First_Name'] [Field:'Last_Name'] - [Field:'Company']<br>
[/Inline]

[/Cache]

→ John Doe - Blue World
   Jane Doe - Blue World
```

To cache content with no expiration:

Use the [Cache] ... [/Cache] tags without the -Expires parameter. The example below shows a cached data source query that never expires. This means that the first result set out put by the contained [Inline] ... [/Inline] tags will be the results that are always output.

```
[Cache:
  -Name='Cache_Name']

[Inline: -Database='Contacts', -SQL='Select * from Contacts.People']
  [Field:'First_Name'] [Field:'Last_Name'] - [Field:'Company']<br>
[/Inline]
```

```
[/Cache]
```

```
→ John Doe - Blue World
   Jane Doe - Blue World
```

To cache content to a session instead of a global variable:

Use the [Cache] ... [/Cache] tags with the -Session parameter. This stores the cached data in a session instead of a global variable, which means that the cached data will expire when the session expires. In the example below, a [Date] tag cached will expire in three hours when the session expires.

```
[Session_Start: -Name='Session_Name', -Expires=3600]
```

```
[Cache:
  -Name='Cache_Name',
  -Session='Session_Name']
```

```
[Date]
```

```
[/Cache]
```

```
→ 9/29/2003 19:13:00
```

To cache content to both a session and a global variable:

Use the [Cache] ... [/Cache] tags with both the -Session and -UseGlobal parameters. This will store the data in both a session and a global variable for maximum control over the cache. In the example below, a [Date] tag cached will never expire unless both the cache is dumped in Lasso Administration and the end-user deletes their session cookie..

```
[Session_Start: -Name='Session_Name']
```

```
[Cache:
  -Name='Cache_Name',
  -Session='Session_Name',
  -UseGlobal]
```

```
[Date]
```

```
[/Cache]
```

```
→ 9/29/2003 19:13:00
```

To conditionally refresh a page:

Use the [Cache] ... [/Cache] tags with the -Condition parameter. The example below conditionally refreshes the cache if the value of [Action_Param:'Cache'] is equal to Yes.

```
[Cache:
-Name='Cache_Name',
-Condition=((Action_Param:'Cache') == 'Yes')]

[Date]

[/Cache]

→ 9/29/2003 21:57:00
```

To create a secure cache:

Use the [Cache] ... [/Cache] tags with the optional -Key parameter. The example below creates a cache named Cache_Name with a key value of password. Only subsequent cache tags containing the parameter -Key='password' will be able to access this cache.

```
[Cache:
-Name='Cache_Name',
-Expires=3600,
-Key='password']

[Inline: -Database='Contacts', -SQL='Select * from people where ID < 3']
[Field:'First_Name'] [Field:'Last_Name'] - [Field:'Company']<br>
[/Inline]

[/Cache]
```

Caching LDML Objects

LDML 7 also includes tags that can cache LDML variables directly without having to use a container tag. When these tags are used, all instances of the LDML variable will be replaced by its cached value until the cache expires or is dumped.

Table 10: LDML Object Cache Tags

Tag	Description
[Cache_Object]	Caches the value of a named LDML variable. Uses the same parameters as the [Cache] ... [/Cache] tags, but requires a -Content parameter that specifies the name of an LDML object variable to be cached. This tag always returns the object value to the page.
[Cache_Store]	Works the same as [Cache_Object], except it does not return a value to the page. Also, the optional -Conditional parameter cannot be used with [Cache_Store].

To cache an LDML object and return its value to the page:

Use the [Cache_Object] tag. The example below adds a variable named Data to the cache. Whenever the Data variable is called while the cache is not expired, any instance of the Data variable will be replaced with its cached value.

```
[Var:'Data'='This is some data']
[Cache_Object: -Name='Cache_Name', -Expires=3600, -Content=$Data]
→ This is some data
```

To cache an LDML object without returning its value to the page:

Use the [Cache_Store] tag. The example below adds a variable named Data to the cache. Whenever the Data variable is called while the cache is not expired, any instance of the Data variable will be replaced with its cached value.

```
[Var:'Data'='This is some data']
[Cache_Store: -Name='Cache_Name', -Expires=3600, -Content=$Data]
```

Cache Control Tags

Additional cache control tags allow values stored in caches to be programmatically fetched and emptied. These tags are described in *Table 11: Cache Control Tags*.

Table 11: Cache Control Tags

Tag	Description
[Cache_Fetch]	Outputs the contents of a Lasso cache. Requires a -Name parameter, which specifies the name of the cache. An optional -Session parameter specifies the session that contains the cache, if applicable.
[Cache_Empty]	Clears a specified cache. The cached contents will be forced to reload at the next page load. Requires a -Name parameter which specifies the name of the cache. An optional -Session parameter specifies the session that contains the cache, if applicable.

To return the contents of a cache:

Use the [Cache_Fetch] tag, where the name of the cache to return is specified in the -Name parameter. The example below returns the value of a cached [Date] tag in a cache named Cache_Name.

```
[Cache_Fetch: -Name='Cache_Name']
```

→ 09/29/2003 19:13:00

To empty a cache:

Use the `[Cache_Empty]` tag, where the name of the cache to empty is specified in the `-Name` parameter.

```
[Cache_Empty: -Name='Cache_Name']
```

Controlling Caches in Lasso Administration

The Lasso global administrator has global control over all caches stored on the Lasso Professional 7 server. The *Utility > Cache* section of Lasso Administration provides information about all current caches, allows caches to be reset, and allows preferences for the caching mechanism to be set.

For more information, see *Chapter 9: Administration Utilities* in the Lasso Professional 7 Setup Guide.

Note: Caches stored in sessions are not visible in Lasso Administration and do not maintain statistics.

Server Push

The `[Server_Push]` tag can be used to progressively download HTML content to a client that supports progressive downloads. All data in the format file up until the location of the `[Server_Push]` tag is sent to the client, but processing of the page continues normally. Multiple `[Server_Push]` tags can be used to send a page to a client in as many segments as desired.

Note: Some Web servers do not support `[Server_Push]`. These Web servers buffer all output from Lasso and stream it to the Web clients themselves.

Most Web browsers will accept progressive downloads only if the `[Server_Push]` tag is placed outside of any HTML container tags (except for `<html> ... </html>` and `<body> ... </body>`). In particular, the `[Server_Push]` tag should not be used within `<table> ... </table>` tags.

Lasso buffers the output of container tags such as `[Loop] ... [/Loop]`, `[If] ... [Else] ... [/If]` and `[Records] ... [/Records]`. The `[Server_Push]` tag can only be used outside of any container tags.

Warning: The `[Server_Push]` tag is incompatible with the `[Header] ... [/Header]`, `[Content_Type]`, `[Redirect_URL]`, `[Cookie_Set]`, and `[Session_Start: -UseCookie]` tags. These tags should not be used on pages which are being sent progressively using `[Server_Push]`.

Table 12: Server Push Tag

Tag	Description
[Server_Push]	Instructs Lasso to send as much of the current format file to the client as possible.

To progressively download a page:

Use the [Server_Push] tag to send sections of the page as they are finished processing. The following example uses a [Server_Push] to force the first part of the page to download, then performs a search using [Inline] ... [/Inline] tags. The header of the page should be visible while the search completes.

```
<h2>Search Results</h2>
[Server_Push]
[Inline: -Database='Contacts',
  -Table='People',
  -KeyField='ID',
  -FindAll]
[Records]
  <br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

Header Tags

The header tags allow the contents of the HTTP response header to be modified before the results of the current format file are served to the visitor. In addition to the tags described in *Table 13: Header Tags*, the [Cookie_Set] tag, [Redirect_URL] tag, and [Server_Push] tag also alter the HTTP response header.

The [Content_Type] and [Header] ... [/Header] tags should be included as the first tags in a format file whenever possible. This ensures that any additional tags that modify the HTTP response header will modify the header defined by these tags.

Lasso uses the character set specified in the [Content_Type] tag to determine how to encode the results of processing a format file before transmitting them to the client's Web browser. By default Lasso will transmit all results in the Unicode single-byte standard UTF-8. See below for examples of how to set the character set to something different.

Table 13: Header Tags

Tag	Description
[Content_Type]	Sets the MIME type of the current HTTP response.
[Header] ... [/Header]	Sets the HTTP header of the response to the contents of the container tags.

Content Type

Use the [Content_Type] tag to set the MIME type for a format file and to set the character set which will be used to transmit the results to the client. The client's Web browser will use this content type and character set to determine how to display the returned data to the client. The [Content_Type] tag should be one of the first tags within a format file.

To set the content type of a format file:

- The following example shows how to return HTML data in a format file encoded using UTF-8. This is the default state for the [Content_Type] tag.

[Content_Type: 'text/html; charset=UTF-8']

- The following example shows how to return HTML data using the Latin-1 (ISO 8859-1) character set. Some older browsers or other Web clients may expect data to be in this character set..

[Content_Type: 'text/html; charset=iso-8859-1']

- The following example shows how to return XML data in a format file with the text/xml MIME type and UTF-8 character set. See *Chapter 29: XML* for more information.

[Content_Type: 'text/xml; charset=utf-8']

- The following example shows how to return WML data in a format file with the text/vnd.wap.wml MIME type and UTF-8 character set. This tag is used when serving data to WAP browsers. See *Chapter 28: Wireless Devices* for more information.

[Content_Type: 'text/vnd.wap.wml; charset=utf-8']

Header Tag

If the [Header] ... [/Header] tags are used within a format file, then the HTTP response header will be set to the contents of the tags. This is a low-level tag that should only be used by developers who are familiar with the structure of HTTP response headers.

The [Content_Type], [Redirect_URL], [Server_Push] and [Cookie_Set] tags can all be used to modify portions of the HTTP response header. These tags are the preferred method for modifying the portions of the header that they affect.

Rules for use of the [Header] ... [/Header] tags:

- The literal string HTTP must appear within the [Header] ... [/Header] tags. Everything before this literal string will be removed from the HTTP response header.
- The first line of a header is a status line that has the following form, HTTP/Version Status_Code Status_Message. For example a soft redirect using the HTTP/1.0 standard is specified as follows.
HTTP/1.0 302 FOUND
- Carriage returns within the [Header] ... [/Header] tags will be replaced by carriage return/line feed pairs.
- All [Header] ... [/Header] tags must end with an empty line. No empty lines are allowed within the [Header] ... [/Header] tags except for the required last line.
- No spaces are allowed at the start of any line within the [Header] ... [/Header] tags.

Headers set using the [Header] ... [/Header] tags must follow the standards defined by the World Wide Web Consortium. Please see their documentation of the HTTP standard for more information.

<http://www.w3c.org/>

To redirect a user to another URL:

Use the [Header] ... [/Header] tags. The following header will redirect the client to the URL specified on the Location line. The URI line is included for compatibility with older browsers. Notice that the destination is URI, not URL.

```
[Header]
HTTP/1.0 302 FOUND
Location: http://www.example.com/default.lasso
URI: http://www.example.com/default.lasso
Server: Lasso Professional 7

[/Header]
```

Note: The [Redirect_URL] tag, documented earlier in this chapter, can also be used to redirect a visitor to a different URL.

To submit a form without reloading the page:

Use the [Header] ... [/Header] tags with a 204 partial content response. The 204 response instructs the client's Web browser that an action has been taken, but that the current rendered page should not be altered. The user can then enter another item into the form and submit it again.

```
[Header]
HTTP/1.0 204
Server: Lasso Professional 7

[/Header]
```

To request authentication information from a client:

Use the [Header] ... [/Header] tags with a 401 unauthorized response. The 401 response instructs the client's Web browser that authentication is required to access the desired resource. The WWW-Authenticate line in the header names the realm which the user is attempting to access so subsequent requests for authentication information will properly retrieve stored passwords as defined by the features of the client's Web browser. The following code asks for authentication for a realm named Example.

```
[Header]
HTTP/1.0 401
WWW-Authenticate: Basic realm="Example"
Server: Lasso Professional 7

[/Header]
```

The first time a client's Web browser receives this response it will check for a stored password or prompt the client to enter a username and password for the specified realm. If the client's Web browser receives the same response again (or sometimes after several authentication attempts) it will assume that the user is not authorized to access the page in question.

Note: See the *Authentication Tags* section of *Chapter 22: Control Tags* for information about LDML tags that automatically prompt for authentication information.

Request Tags

Lasso includes a number of tags that return information about the current HTTP request. These tags can be used to inspect the URL, GET arguments, POST arguments, form method, or even the raw HTTP request. These tags are summarized in *Table 14: Request Tags*.

Table 14: Request Tags

Tag	Description
[Client_ContentLength]	Returns the length in characters of the current POST parameters.
[Client_ContentType]	Returns the MIME type requested by the current HTTP request.
[Client_FormMethod]	Returns the method used to load the current page, either GET or POST.
[Client_GETArgs]	Returns a string containing all the arguments passed along with the URL in the current request.
[Client_GETParams]	Returns a pair array containing an element for each parameter passed along with the URL in the current request.
[Client_Headers]	Returns the text of the HTTP request which called this page.
[Client_Password]	Returns the password specified by the current client.
[Client_POSTArgs]	Returns a string containing all the arguments passed along with the URL as a POST parameter in the current request.
[Client_POSTParams]	Returns a pair array containing an element for each parameter passed along with the URL as a POST parameter in the current request.
[Client_Username]	Returns the username specified by the current client.
[Response_FilePath]	Returns the path to the file which is being served from the Web server root.
[Response_LocalPath]	Returns the path to the Web server root.
[Response_Path]	Returns the folder from which the current file is being served relative to the Web server root.
[Response_Realm]	Returns the name of the current realm reported by the Web server.

To provide a link to the current Web page:

The [Response_FilePath] tag can be used to provide a link that reloads the current Web page. The following example provides a simple link that reloads the current Web page without any GET or POST parameters.

```
<a href="[Response_FilePath]"> Reload this page </a>
```

To display the current GET parameters:

Use the [Loop] ... [/Loop] tags and the [Array->Get] tag to loop through the [Client_GetParams] array. The results are shown for the following URL:
<http://www.example.com/default.lasso?name1=value1&name2=value2>.

```
[Loop: (Client_GetParams)->Size]
  [Variable: 'GET_Variable' = (Client_GetParams)->(Get: (Loop_Count))]
  <br>[Output: $GET_Variable->First] = [Output: $GET_Variable->Second]
[/Loop]
```

→
name1 = value1

name2 = value2

The same methodology can be used for the output of the [Client_PostParams] tag.

To accept only POST parameters:

Check the [Client_FormMethod] tag to see whether it equals GET or POST. If it is not set to the desired value then redirect the client to another page. The following code redirects the user to *error.lasso* if the current page is not loaded with POST parameters.

```
[If: (Client_FormMethod) != 'POST']
  [Redirect_URL: 'http://www.example.com/error.lasso']
[/If]
```

Note: It is possible to load a page with both POST and GET parameters so a complete solution needs to check that a POST form method was used and scan the GET parameters.

Client Tags

Lasso includes a number of tags that return information about the current client including what type of browser they are using and where their client machine is located. These tags are summarized in *Table 15: Client Tags*.

Table 15: Client Tags

Tag	Description
[Client_Address]	Returns the host name of the current client.
[Client_Browser]	Returns the type of browser used by the current client.
[Client_IP]	Returns the IP address of the current client.
[Client_Type]	Returns the type of browser used by the current client.

Note: Lasso also includes a set of [WAP_...] tags that return information about clients using WAP browsers. See *Chapter 28: Wireless Devices* for more information.

To check whether a client is using a specific browser:

The [Client_Browser] tag can be used to return the type of browser the client is using. The following example checks whether the browser type contains Netscape and displays an appropriate message if it does.

```
[If: (Client_Browser) >> 'Netscape']
  <br>You are using a supported Netscape browser.
[Else]
  <br>You are using an unsupported browser of type: [Client_Browser].
[/If]
```

Server Tags

Lasso provides a number of tags which return information about the current Web server. The information returned by the tags in **Table 16: Server Tags** can be used to determine whether a page is being served normally or securely or to output information to log files.

Table 16: Server Tags

Tag	Description
[Server_Name]	Returns the name of the current server.
[Server_Port]	Returns the port which the current request is being served. Usually 80 for normal HTTP requests or 443 for secure HTTPS requests.

To check whether a page is being served securely:

Check the output of the [Server_Port] tag. Most Web servers serve normal HTTP traffic on port 80 and secure, SSL encrypted HTTPS traffic on port 443. The following example displays a reassuring message if the page is being served securely or a warning if the page is not being served securely.

```
[If: (Server_Port) == 80]
  <p><font color="red">Warning: this page is not being served securely.</font>
[Else: (Server_Port) == 443]
  <p><font color="blue">Don't panic: this page was served securely.</font>
[Else]
  <p><font color="yellow">Caution: this page is served from an unknown port.</font>
[/If]
```

To log information about server requests to a log file:

Use the [Server_...] and [Client_...] tags to return information about the current visitor and what page they are visiting. The following code will log the current date and time, the visitor's IP address, the name of the server

and the page they were loading, and the GET and POST parameters that were specified.

```
[Log: 'E://Logs/LassoLog.txt']  
[Date]  
[Client_IP] [Server_Name] [Response_FilePath]  
[Client_GETArgs] [Client_POSTArgs]  
[/Log]
```

See *Chapter 20: Files and Logging* for more information about the [Log] ... [/Log] tags.

28

Chapter 28

Wireless Devices

This chapter describes how to create pages in the Wireless Markup Language (WML) which can be served to clients using Wireless Application Protocol (WAP) browsers.

- *Overview* introduces wireless devices.
- *Formatting WML* describes how to specify the MIME type and encode data for wireless browsers.
- *WAP Tags* describes the tags in LDML that allow the characteristics of a WAP client to be returned.
- *WML Example* shows how to create a page which a WAP client can use to search a database and retrieve the results.

Overview

Lasso provides support for serving data to cellular phones and personal digital assistants that support the Wireless Application Protocol (WAP) and the Wireless Markup Language (WML). Serving data to WAP devices (e.g. WAP browsers) is conceptually the same as serving pages to Web browsers, but there are some special considerations that need to be taken into account.

WAP devices require pages to be formatted using the XML-based Wireless Markup Language. Documentation of this language is beyond the scope of this manual. Please consult a book on WAP/WML for more information about how to create pages in WML. Since WML is based on XML, many XML books also contain information about WML.

Lasso does not serve pages to WAP browsers directly. Instead, most WAP browsers communicate with a gateway that contacts Lasso for WML pages

and images. The gateway is responsible for performing some manipulation of WML pages and images to ensure they are formatted properly for the WAP browser. A WML-based Web site built using the tags described in this chapter will produce code that is very friendly to the gateway and ensures high fidelity of the site when it is viewed using a WAP browser.

Note: Since WML is based on XML all of the techniques in the following chapter on XML can be used on WML content.

Formatting WML

WAP browsers require pages to be sent using the MIME type of `text/vnd.wap.wml` and a UTF-8 character set. The `[Content_Type]` tag can be used to set the MIME type and character set of a page served by Lasso. This tag simply adjusts the header of the page served by Lasso, it does not perform any conversion of the data on the page.

To specify a format file contains WML:

Use the following tag as the very first line of any files which will be served to WAP browsers. Notice that the tag accepts only a single parameter, the `charset` argument which is appended to the MIME type argument with a semi-colon ;.

```
[Content_Type: 'text/vnd.wap.wml; charset=utf-8']
```

To serve WML:

WML data can be served using the `[XML_Serve]` tag with the optional `-Type` parameter set to `text/vnd.wap.wml`. When the `[XML_Serve]` tag is used all processing of the current page halts and the parameter of the tag is returned as the contents of the page. This is useful to prevent any stray comments or characters from being sent to WML browsers.

The following example serves some simple WML data in place of the current format file. No tags after the `[XML_Serve]` tag will be processed.

```
[Variable: 'WMLData' = '<?xml version="1.0" encoding="utf-8" ?>
<wml>
  <card name="card_one">
    <p><b>Hello WAP user!</b></p>
  </card>
</wml>']

[XML_Serve: $WMLData, -Type='text/vnd.wap.wml']
```

To format WML:

The data served by Lasso should be formatted using WML. Most WML pages have the following format, an `<?XML ... ?>` declaration followed by `<wml> ... </wml>` tags that surround one or more `<card> ... </card>` tags. The contents of the `<card> ... </card>` tags are formatted like tiny HTML pages. The following example shows a WML file with a single card.

```
[Content_Type: 'text/vnd.wap.wml; charset=utf-8']
<?xml version="1.0" encoding="utf-8" ?>
<wml>
  <card name="card_one">
    <p><b>Hello WAP user!</b></p>
  </card>
</wml>
```

Most HTML text-formatting tags can be used to format WML pages although the actual set of tags supported may differ from browser to browser. Tables can be used to format data into columns. All tags in WML have an opening and a closing tag. All paragraph tags `<p> ... </p>` must be closed. A tag which opens and then closes immediately can be written with a slash before the trailing angle bracket, `
</br>` can be written `
`.

Every parameter of a tag must have a value. For example, the `<input>` tag for a check box takes a parameter `checked=""` rather than the simple `checked` parameter which HTML allows.

```
<input type="checkbox" name="Field_Name" value="Value" checked="">
```

To specify WML links:

Links can be included using the anchor convention to link to cards within the same document or a different document. The following code would create a link to the card defined above if it were inserted into another card in the same document.

```
<a href="#card_one"> Link to card one </a>
```

If the card defined above was saved in a document named `default_wml.lasso` then the following link inserted into a card in another document would link directly to it. Both the name of the document and the name of the card are included in the link.

```
<a href="default_wml.lasso#card_one"> Link to card one </a>
```

To specify WML forms:

Forms can be included in WML documents using most of the form input tags. Since WAP browser screens are usually very small, only a few form elements can usually be shown on screen at the same time. Also, since most WAP browsers have limited text capabilities it is often desirable to

place options in `<select> ... </select>` tags rather than having the client type them in. The following code shows a form that contains a single button. When the form is submitted, the card `Card_One` in `default_wml.lasso` is returned as the result.

```
<form action="default_wml.lasso#card_one" method="POST">
  <p><input type="submit" name="-Nothing" value="Submit Form"></p>
</form>
```

To encode data for WML:

The data displayed in WML pages should be XML encoded. The `[Encode_Set] ... [/Encode_Set]` tags can be used to change the default encoding for all substitution tags in an entire WML page. The following example shows a WML page with an enclosing set of `[Encode_Set] ... [/Encode_Set]` tags. The value of the `[Variable]` tag will be XML encoded, ensuring that it displays properly in a WAP browser.

```
[Content_Type: 'text/vnd.wap.wml; charset=utf-8']
<?xml version="1.0" encoding="utf-8">
[Encode_Set: -EncodeXML]
  <wml>
    <card name="card_one">
      <p>[Variable: 'WML_Data']</p>
    </card>
  </wml>
[/Encode_Set]
```

Tags which return XML tags should not have their values encoded. Tags which return XML data require an `-EncodeNone` encoding keyword in order to ensure that the angle brackets and other markup characters are not encoded into XML entities. The following example shows a variable that returns an entire `<card> ... </card>`. The `[Variable]` tag has an `-EncodeNone` keyword so the angle brackets within the WML data are not encoded.

```
[Content_Type: 'text/vnd.wap.wml; charset=utf-8']
[Variable: 'WML_Data' = '<card name="card_one"><p>Hello WAP user!</card>']
<?xml version="1.0" encoding="utf-8">
[Encode_Set: -EncodeXML]
  <wml>
    [Variable: 'WML_Data', -EncodeNone]
  </wml>
[/Encode_Set]
```

WAP Tags

LDML 7 includes a set of tags that return information about WAP clients. These tags allow a Lasso developer to determine if the current client is using a WAP browser and to determine the size of the screen and how many buttons the browser supports.

The tags are summarized in *Table 1: WAP Tags*. None of the tags return a value if the current client is not using a WAP browser or if the WAP browser does not report the appropriate information in their WAP request. The [WAP_IsEnabled] tag should always be used first to determine if the client is a WAP browser before the other tags are used.

Table 1: WAP Tags

Tag	Description
[WAP_IsEnabled]	Returns True if the current client is using a WAP enabled browser.
[WAP_MaxButtons]	Returns the number of buttons supported by the current client's WAP browser.
[WAP_MaxColumns]	Returns the number of text columns in the screen of the current client's WAP browser.
[WAP_MaxHorzPixels]	Returns the width of the screen in pixels of the current client's WAP browser.
[WAP_MaxRows]	Returns the number of text lines in the screen of the current client's WAP browser.
[WAP_MaxVertPixels]	Returns the height of the screen in pixels of the current client's WAP browser.
[XML_Serve]	Returns WML data in place of the current format file. The first parameter is the WML data to be served. -Type parameter should be set to text/vnd.wap.wml

To display a different page if a client is WAP enabled:

Use the [WAP_IsEnabled] tag to check whether a client is using a WAP browser or not. The following code returns the file default_wml.lasso if the user is using a WAP browser or the file default_html.lasso if they are using a normal Web browser.

```
[If: (WAP_IsEnabled)]
  [Content_Type: 'text/vnd.wap.wml; charset=utf-8']
  [Include: 'default_wml.lasso']
[Else]
  [Include: 'default_html.lasso']
[/If]
```

To choose a graphic based on the size of a WAP browser screen:

Use the [WAP_MaxHorzPixels] and [WAP_MaxVertPixels] tags to determine the size of the client's screen. The following example displays a different graphic if the client's screen is less than 72 pixels in height or width, if it is less than 144 pixels in height or width, or if it is larger.

```
[if: (WAP_MaxHorzPixels) <= 72 || (WAP_MaxVertPixels) <= 72]
  
[Else: (WAP_MaxHorzPixels) <= 144 || (WAP_MaxVertPixels) <= 144]
  
[Else]
  
[/If]
```

WML Example

The following example shows how to create a page that allows a client to search a database through a WAP browser. The client will be able to search a database named Contacts for either the First_Name or Last_Name and will receive a list of Phone_Numbers in response.

The example is given first in a square bracket version using marked up WML code. The second version uses LassoScript and the [XML_Serve] tag to serve programmatically created WML.

Square Bracket Version

The initial page default.lasso includes a check to see whether the client is using a WAP browser or not. If they are not using a WAP browser then they are forwarded to an error page using the [Redirect_URL] tag.

```
[If: (WAP_IsEnabled) == False]
  [Redirect_URL: 'error.lasso']
[/If]
```

The remainder of the initial page is a card called form that contains an HTML form which allows the user to search the database for either a First_Name or a Last_Name. When the form is submitted the results card of response.lasso is returned.

```
[Content_Type: 'text/vnd.wap.wml; charset=utf-8']
<?xml version="1.0" encoding="utf-8">
[Encode_Set: -EncodeXML]
<wml>
  <card name="form">
    <form action="response.lasso#results" method="POST">
      First: <input type="text" name="First_Name" value=""/>
```



```

        <br/>Last: <input type="text" name="Last_Name" value=""/>
        <br/><input type="submit" name="-Nothing" value="Submit"/>
    </form>
</card>
</wml>
[/Encode_Set]

```

The results page response.lasso contains an [Inline] ... [/Inline] that performs the actual search. It retrieves the values for First_Name and Last_Name using [Action_Param] tags. The search results are sorted first by Last_Name, then by First_Name. None of the [Field] tags require encoding keywords since the default encoding for the page is set to XML encoding using [Encode_Set] ... [/Encode_Set] tags. An error message is returned if no records are found. A link is provided to return to the search page default.lasso so a new search can be performed.

```

[Content_Type: 'text/vnd.wap.wml; charset=utf-8']
<?xml version="1.0" encoding="utf-8">
[Encode_Set: -EncodeXML]
<wml>
    <card name="results">
        [Inline: -Database='Contacts',
        -Table='People',
        -KeyField='ID',
        'First_Name' = (Action_Param: 'First_Name'),
        'Last_Name' = (Action_Param: 'Last_Name'),
        -SortField='Last_Name',
        -SortField='First_Name',
        -Search]
        [If: (Found_Count) <= 0]
            <br/>No phone numbers were found.
        [/If]
        [Records]
            <br/>[Field: 'First_Name'] [Field: 'Last_Name'] [Field: 'Phone_Number']
        [/Records]
        [/Inline]
        <br/><a href="default.lasso#form"> Search Again </a>
    </card>
</wml>
[/Encode_Set]

```

LassoScript Version

The initial page default.lasso includes a check to see whether the client is using a WAP browser or not. If they are not using a WAP browser then they are forwarded to an error page using the [Redirect_URL] tag.

```

<?LassoScript
  If: (WAP_IsEnabled) == False;
    Redirect_URL: 'error.lasso';
  /If;
?>

```

The remainder of the initial page is a card called form that contains an HTML form which allows the user to search the database for either a First_Name or a Last_Name. When the form is submitted the results card of response.lasso is returned.

```

<?LassoScript
  Variable: 'WML_Content' = <?xml version="1.0" encoding="utf-8">
  $WML_Content += '<wml><card name="form">';
  $WML_Content += '<form action="response.lasso#results" method="POST">';
  $WML_Content += 'First: <input type="text" name="First_Name" value=""/>';
  $WML_Content += '<br/>Last: <input type="text" name="Last_Name" value=""/>';
  $WML_Content += '<br/><input type="submit" name="-Nothing" value="Submit"/>';
  $WML_Content += '</form></card></wml>';

  XML_Serve: $WML_Content, -Type='text/vnd.wap.wml';
?>

```

The results page response.lasso contains an [Inline] ... [/Inline] that performs the actual search. The actual response is collected in the WML_Content variable. The [Field] tags have encoding explicitly set.

```

<?LassoScript
  Variable: 'WML_Content' = <?xml version="1.0" encoding="utf-8">';
  $WML_Content += '<wml><card name="results">';
  Inline: -Search, -Database='Contacts', -Table='People', -KeyField='ID',
    'First_Name' = (Action_Param: 'First_Name'),
    'Last_Name' = (Action_Param: 'Last_Name'),
    -SortField='Last_Name', -SortField='First_Name';
  If: (Found_Count) <= 0;
    $WML_Content += '<br/>No phone numbers were found.';
  /If;
  Records;
    $WML_Content += '<br/>' + (Field: 'First_Name', -EncodeXML) + ' ';
    $WML_Content += (Field: 'Last_Name', -EncodeXML) + ' ';
    $WML_Content += (Field: 'Phone_Number', -EncodeXML);
  /Records;
  /Inline;
  $WML_Content += '<br/><a href="default.lasso#form"> Search Again </a>';
  $WML_Content += '</card></wml>';

  XML_Serve: $WML_Content, -Type='text/vnd.wap.wml';
?>

```

29

Chapter 29

XML

This chapter describes how to parse and create Extensible Markup Language (XML) data and how to communicate using XML Remote Procedure Calls (XML-RPC).

- *Overview* introduces Lasso's XML support.
- *XML Glossary* introduces XML specific terms.
- *XML Data Type* describes how to parse and create XML data using the XML data type.
- *XPath Extraction* describes how to use XPath parameters to extract specific data from an XML file.
- *XSLT Style Sheet Transforms* describes how to transform XML data using XSLT style sheets.
- *XML Stream Data Type* describes how to parse XML documents using a stream model similar to a SAX parser.
- *XML-RPC* describes how to send and process remote procedure calls for communication between servers.
- *SOAP* describes how to send and process SOAP remote procedure calls for communication between servers.
- *Serving XML* describes how to serve XML data in place of the current format file.
- *Formatting XML* describes how to specify the MIME type and encode data for XML clients.
- *XML Templates* describes the XML templates included with Lasso and how to use them to format database action results as XML data.

Overview

Lasso provides support for a number of different XML standards which make parsing, validating, creating, transforming, and serving XML easy.

Lasso includes an XML data type that automatically parses XML from string values. The XML data type represents XML data as a tree data structure and includes member tags for manipulating the individual tags which make up the XML data. Changes can be made to the XML data type and will be automatically converted to proper XML syntax when output to the Web browser.

Lasso can validate XML data according to a Document Type Definition (DTD). If the XML data does not correspond to the structure defined by the DTD then an error will be returned to the user.

Lasso supports automatic transformations of XML data using the XSLT style sheets. An XSLT transform can be applied to XML data stored in a variable, database field, or file. In addition, a single format file can be repurposed for many different clients through the use of a stylesheet transform just prior to serving.

Lasso supports extracting individual XML elements from XML data using XPath parameters. The XPath language complements Lasso's built-in XML data type allowing sophisticated queries on XML data. The `[XML_Extract]` tag can be used to work with large XML documents.

XML-RPC support allows Lasso to communicate between servers. Lasso supports incoming XML-RPC requests through custom XML-RPC tags that are automatically processed or allows incoming requests to be processed by any format file. XML-RPC requests can be easily generated and sent to other servers on the Internet for processing.

Finally, Lasso can serve XML data which conforms to any Document Type Definition (DTD) or XML Schema using the same tools which allow Lasso to serve any style of HTML, WML, or other browser-based languages.

XML data needs to be formatted according to the rules defined by the World Wide Web Consortium. Documentation of this language is beyond the scope of this manual. Please consult a book on XML for more information about how to create properly formatted XML data.

Note: The XML data type should not generally be used to process XML documents larger than about 3 megabytes depending on their complexity. The `[XML_Extract]` tag can be used to parse much larger XML documents and extract specific elements for further processing.

XML Glossary

Here is a short glossary of essential terms which will help you understand the rest of this documentation if you are new to XML.

- **HTML** – HyperText Markup Language (HTML) is the language in which the World Wide Web is formatted and is characterized by markup tags enclosed in angle brackets. HTML is a subset of SGML.
- **XML** – Extensible Markup Language (XML) is the universal format for structured documents and data on the Web. XML is a subset of SGML.
- **SGML** – Standard Generalized Markup Language (SGML) is a system for defining markup languages. Authors mark up their documents by representing structural, presentational, and semantic information alongside content. HTML and XML are both based on SGML.
- **DTD** – A Document Type Definition (DTD) is a type of file associated with SGML and XML documents that defines how the markup tags should be interpreted by the application presenting the document.
- **Schema** – An XML-based method of specifying the structure of an XML document. Basically, a replacement for a DTD, but specified in XML syntax. This is an emerging standard which is yet to be ratified by the World Wide Web Consortium (W3C) at the time of this writing.
- **XPath** – A language which is used to define the location of one or more tags or attributes within XML data. XPaths can be used to extract tags or attributes from XML data and are used in XSLT style sheets. XPaths are used to extract specific elements from a larger XML document.
- **XSL** – Extensible Stylesheet Language (XSL) is a language for expressing stylesheets. An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.
- **XSLT** – XSL Transformations (XSLT) is a language for transforming XML documents into other XML documents. XSLT is designed for use as part of XSL, which is a stylesheet language for XML.
- **XML-RPC** – XML Remote Procedure Call (XML-RPC) allows actions to be performed on another server on the Internet and for data to be returned.
- **WML** – Wireless Markup Language (WML) is an XML-based language in which “cards” for display on cellular phones and other wireless devices are created.

XML Data Type

The XML data type in Lasso automatically parses XML data which is stored in a variable. The member tags of the XML data type can then be used to inspect and change the XML data. *Table 1: XML Data Type Tag* describes the tag which is used to convert string data to the XML data type.

Lasso also provides an alternate method of parsing XML data that may be more efficient for very large XML documents. This method is described in the *XML Stream Data Type* section below.

Table 1: XML Data Type Tag

Tag	Description
[XML]	Requires a single parameter which is a string containing validly formatted XML data. Optional -DTD parameter specifies a DTD against which the XML should be validated or optional -Schema parameter specifies an XML schema against which the XML should be validated. Optional -Validation parameter specifies whether the validation errors should be reported. Proper values include 'always', 'never', and 'auto'. The default is 'auto' which reports errors only if a schema or DTD was specified. Optional -Namespaces parameter specifies whether XML namespaces should be processed. Defaults to false. Optional -FullCheck parameter specifies whether full schema checking should be performed. This check can be very time consuming. Defaults to false.

XML data from any source can be parsed and manipulated using Lasso by first storing the XML data in a variable and then casting it to the XML data type using the [XML] tag. Lasso can work with XML data from a database field, XML file, remote Web application server, XML-RPC request, FTP site, etc. Or, Lasso can work with XML data that is created programmatically within a variable.

To parse XML data:

Use the [XML] tag to cast a string variable to the XML data type and parse the XML data which is contained within the variable. The following example stores a string of XML data in a variable, then casts it to XML.

```
[Variable: 'XML_String' = '<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <RECORD>
    <FIELD name="First Name">John</FIELD>
    <FIELD name="Last Name">Doe</FIELD>
  </RECORD>
</ROOT>']
[Variable: 'XML_Data' = (XML: $XML_String)]
```

The variable XML_Data now contains a parsed representation of the data from XML_String. If the variable XML_Data is output the value of XML_String will simply be returned, but if the type of the variable is checked it will be XML.

```
[Variable: 'XML_Data']
<br>Type: [Output: $XML_Data->Type]

→ <?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <RECORD>
    <FIELD name="First Name">John</FIELD>
    <FIELD name="Last Name">Doe</FIELD>
  </RECORD>
</ROOT>
<br>Type: XML
```

The parts of the parsed XML data can be accessed using the member tags of the XML data type which are detailed in *Table 2: XML Member Tags*.

Table 2: XML Member Tags

Tag	Description
[XML->Attributes]	An array of pairs for each of the attributes of the root tag.
[XML->Name]	The name of the root tag.
[XML->Children]	An array of XML objects for each of the children tags of the root tag.
[XML->Contents]	The contents of the root tag.
[XML->Document]	Returns the root tag of the current XML document.
[XML->Extract]	Returns the value for an XPath. Requires a single parameter which is the XPath to be evaluated. Returns different values depending on the XPath.
[XML->ExtractOne]	Works the same as [XML->Extract] but only returns the first element found by the XPath.

[XML->NameSpaces]	Returns an array of namespaces for the namespaces declared for this node. The array has pairs in which the first element is the namespace prefix and the second element is the URI of the namespace.
[XML->NextSibling]	Returns the next sibling of the current node.
[XML->NodeType]	Returns the type of the current node.
[XML->Parent]	Returns the parent for the current node.
[XML->Path]	Returns the path to the current node from the root of the document.
[XML->PreviousSibling]	Returns the previous sibling for the current node.
[XML->Transform]	Performs an XSLT style sheet transformation on the current XML object. Requires a string which contains a valid XSLT style sheet. Returns a new XML object with the results of the transformation.

These member tags can be used to inspect and modify the attributes and children of an XML tag.

To find specific children of an XML tag:

Use the [XML->Children] tag to get an array of children of an XML tag. For example, the children of the <ROOT> tag in XML_Data can be returned as follows. The result is always an array even if there is only one child of the root XML tag.

```
[XML_Data->Children]
```

→ (Array: (<RECORD> ... </RECORD>))

The children of the <RECORD> tag can be found by extracting the <RECORD> tag from the array of children using [Array->Get] and then using [XML->Children] to return an array of <FIELD> tags..

```
[Variable: 'XML_Record' = XML_Data->Children->(Get: 1)]
[XML_Record->Children]
```

→ (Array: (<FIELD name="First_Name">John</FIELD>),
(<FIELD name="Last_Name">Doe</FIELD>))

To display the attributes of an XML tag:

Use the [XML->Attributes] tag. The following example returns the attributes of the first element from the XML_Record variable in the previous example. The [Iterate] ... [/Iterate] tags are used to cycle through the array of attributes and the elements of each attribute pair are displayed.


```
[Variable: 'XML_Attributes' = $XML_Record->Attributes]
[Iterate: $XML_Attributes, (Variable: 'Attribute')]
  <br>[$Attribute->First] = [$Attribute->Second]
[/Iterate]
```

→
Name = First_Name

To display the contents of an XML tag:

Use the [XML->Contents] tag. The following example returns the contents of the first element from the XML_Record variable in the example above.

```
[Output: $XML_Record->Contents]
```

→ John

XPath Extraction

XPath is a language that allows XML data to be searched for specific tags or attributes. An XPath expression instructs how to get to a specific tag or tags within XML data similarly to how a file system path instructs how to get to a specific file within a hard drive.

This is the preferred method of processing large XML documents. An XPath can be used to extract the relevant elements from the large XML document and then those individual elements can be converted to the XML data type for further processing.

For example, the Lasso Service application on Mac OS X is represented by the following path. The path says to go to the root of the file system /, enter the Applications folder then the Lasso Professional 7 folder, and look for the file named LassoService.

```
/Applications/Lasso Professional 7/LassoService
```

Similarly, an XPath to navigate through the following XML data can be constructed.

```
[Variable: 'XML_String' = '<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <RECORD>
    <FIELD name="First Name">John</FIELD>
    <FIELD name="Last Name">Doe</FIELD>
  </RECORD>
</ROOT>']
```

The following XPath starts at the root of the XML data, the <ROOT> tag represented by /ROOT. It enters the <RECORD> tag and returns the <FIELD> tag which has a name attribute equal to First_Name.

```
/ROOT/RECORD/FIELD[@name="First_Name"]
```

The [XML_Extract] tag allows an XPath to be applied to XML data within Lasso and for the results to be returned.

Table 3: [XML_Extract] Tag

Tag	Description
[XML_Extract]	Accepts two parameters and returns an array of string. -XML is the XML source data or -File specifies the path to a file that contains the XML source data. -XPath is the XPath that describes what data to return.

Note: The [XML_Extract] tag will read XML data from a -File parameter if it is present. This is the preferred method of working with large XML documents since Lasso can parse the file without reading it into memory. If no -File parameter is specified then the data passed directly to the -XML parameter is used instead.

The XPath from above would be applied to the XML data in this way.

```
[XML_Extract: -XML=$XML_String,  
-XPath="/ROOT/RECORD/FIELD[@name=First_Name]"]
```

The return value is an array containing a single string representing the tag which was found.

→ (Array: (<FIELD name="First_Name">John</FIELD>))

File paths generally only allow inspecting the names of files and directories. XML tags have a name, children, attributes, contents, etc. The XPath allows any of these different aspects of XML tags to be used in specifying a path to a specific tag or set of tags.

Table 4: Simple XPath Expressions includes the basic elements of an XPath. These can be combined with the conditional functions detailed in *Table 5: Conditional XPath Expressions* to create sophisticated queries allowing very specific sets of tags and sub tags to be extracted from XML data.

Note: A full discussion of XPath syntax is beyond the scope of this book. Please consult a book about XML for full details about XPath syntax.

Table 4: Simple XPath Expressions

Expression	Description
/	Selects the root element of the XML data. The first XML tag in the XML data is a child of the root element.
/*	Selects all children elements from the current element including XML tags and text elements. /node() is a synonym.
/tagname	Selects all XML tag children with the specified tag name from the current element.
/text()	Selects all text element children from the current element .
//*	Selects all descendants starting from the current element including XML tags and text elements. //node() is a synonym.
//tagname	Selects all XML tags descendants with the specified tag name starting from the current element.
//text()	Selects all text element descendants starting from the current element.
/@*	Selects all attributes of the current tag.
/@attribute	Selects all attributes of the current tag with the specified attribute name.

These expressions are assembled into a path by placing them in the appropriate order depending on the tags or attributes that need to be extracted. The following are some examples of XPath expressions and what tags they would extract from the XML data specified on the previous page.

- Select all <ROOT> tags.
/ROOT
- Select all <RECORD> tags which are contained in the <ROOT> tag.
/ROOT/RECORD/
- Select all <FIELD> tags which are children of a <ROOT> and <RECORD> tag.
/ROOT/RECORD/FIELD
- Select the text contents of all <FIELD> tags which are children of a <ROOT> and <RECORD> tag.
/ROOT/RECORD/FIELD/text()
- Select all <FIELD> tags no matter what the name of their parent tag was.
//FIELD
- Select the name attributes from all <FIELD> tags.
//FIELD/@name

- Select all attributes from all <FIELD> tags.

```
//FIELD/@*
```

- Select all text elements from the XML data.

```
//text()
```

To extract tags from XML data using simple XPath expressions:

The simple XPath expressions can be used to find a specific set of nodes within XML data. For example, using the same XML data as for the example above the following XPath expressions return the specified results.

```
[Variable: 'XML_String' = '<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <RECORD>
    <FIELD name="First Name">John</FIELD>
    <FIELD name="Last Name">Doe</FIELD>
  </RECORD>
</ROOT>']
```

- The root tag of the XML data and all of its contents can be returned using /ROOT/. The <ROOT> tag and all its contents are returned.

```
[XML_Extract: -XML=$XML_String, -XPath="/ROOT/"]
```

→ (Array: (<ROOT> ... </ROOT>))

- All children of the root tag can be returned using /ROOT/*. The <RECORD> tag and all its contents are returned.

```
[XML_Extract: -XML=$XML_String, -XPath="/ROOT/*"]
```

→ (Array: (<RECORD> ... </RECORD>))

- All <FIELD> tags in the XML data can be returned using //FIELD. The two <FIELD> tags and all of their contents are returned.

```
[XML_Extract: -XML=$XML_String, -XPath="//FIELD"]
```

→ (Array: (<FIELD name="First Name">John</FIELD>),
(<FIELD name="Last Name">Doe</FIELD>))

- The name parameter from all <FIELD> tags in the XML data can be returned using //FIELD/@name. The name parameters of the <FIELD> tags are returned.

```
[XML_Extract: -XML=$XML_String, -XPath="//field/@name"]
```

→ (Array: (First_Name), (Last_Name))

Many complex queries can be created using the simple XPath parameters. In addition, XPath allows for conditional expressions to be used on the simple XPath expressions. These are detailed in *Table 5: Conditional XPath Expressions*.

Table 5: Conditional XPath Expressions

Expression	Description
[n]	A number selects one specific element from an array of returned tags or parameters.
[last()]	Returns the last element from an array of returned tags or parameters.
[tagname]	Returns only tags which have one or more children with the specified tag name.
[@attribute]	Returns only those tags which have the specified attribute.
[@attribute=value]	Returns only those tags which have the specified attribute equal to the value.
[.=value]	Returns only those tags which have their contents equal to the specified value.
[expression = value]	Returns only those tags for which the expression is equal to the specified value. Can also use < <= > >= or != for numeric comparisons.
[starts-with(expression, value)]	Returns only those tags which have an attribute or child tag that starts with the specified value.
[contains(expression, value)]	Returns only those tags which have an attribute or child tag that contains the specified value.
[count(expression)]	Returns the number of elements in an array of returned tags or parameters.

In addition to the expressions detailed in this table it is possible to use numeric functions + - * div mod, boolean operations and or or, and parentheses to create more complex expressions.

To extract tags from XML data using conditional XPath expressions:

The conditional XPath expressions can be used to find a specific set of nodes within XML data. For example, using the same XML data as the example above, the following XPaths return the specified results.

```
[Variable: 'XML_String' = '<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <RECORD>
    <FIELD name="First Name">John</FIELD>
    <FIELD name="Last_Name">Doe</FIELD>
  </RECORD>
</ROOT>']
```

- The first <FIELD> tag in the XML data can be returned using //FIELD[1]. The first <FIELD> tag and all of its contents are returned.
[XML_Extract: -XML=\$XML_String, -XPath="//FIELD[1]"]

→ (Array: (<FIELD name="First_Name">John</FIELD>))

- The last <FIELD> tag descendant of the root tag can be returned using //FIELD[last()]. The second <FIELD> tag and all of its contents are returned.

[XML_Extract: -XML=\$XML_String, -XPath="//FIELD[last()]"]

→ (Array: (<FIELD name="Last_Name">Doe</FIELD>))

- All <FIELD> tag descendants of the root tag which have their contents equal to John can be returned using //FIELD[.="John"]. The . in the expression represents the current tag that is being examined. The first <FIELD> tag and all of its contents are returned.

[XML_Extract: -XML=\$XML_String, -XPath="//FIELD[.="John"]"]

→ (Array: (<FIELD name="First_Name">John</FIELD>))

- All <FIELD> tag descendants of the root tag which have a name parameter that contains the word Name can be returned using //FIELD[contains(@name, "Name")]. Both <FIELD> tags and all of their contents are returned.

[XML_Extract: -XML=\$XML_String, -XPath="//field[contains(@name, "Name")]"]

→ (Array: (<FIELD name="First_Name">John</FIELD>),
(<FIELD name="Last_Name">Doe</FIELD>))

XSLT Style Sheet Transforms

XML style sheets allow one set of XML data to be transformed to a different set. A single base XML document can be converted so it can be used in many different situations. For example, a single document could be converted to HTML for display in a Web browser and to WML for display in a wireless device.

Lasso allows XSLT transforming style sheets to be applied to XML data using the [XML_Transform] tag. The input of the tag is a string containing XML source data and a string containing a valid XSLT style sheet. The result is the string that is generated by applying the style sheet to the data.

Note: A full discussion of XSLT syntax is beyond the scope of this book. Please consult a book about XML for full details about XML style sheets.

Table 6: [XML_Transform] Tag

Tag	Description
[XML_Transform]	Accepts two parameters and returns a string. -XML is the XML source data. -XSL is the XSLT style sheet to be applied.

Note: It is important to include `version` and `xmlns:xsl` parameters in the opening `<xsl:stylesheet>` tag passed to Lasso so the XSLT processor knows what version of XSL is being used and what namespace to use when parsing the XSLT style sheet.

To transform XML data using an XSLT style sheet:

Use the [XML_Transform] tag. The following example uses an XSLT style sheet stored in `XSLT_String` to transform XML data stored in `XML_String` and output an HTML table.

The XSLT style sheet is an XML document that uses XPath expressions to select portions of the XML data and transform them into a different format. In this case, XML data is transformed into HTML for display in a Web browser.

The `<xsl:template>` tag specifies what XML element the style sheet will transform. The `<xsl:for-each>` tags accept an XPath that specifies a specific set of elements to iterate through. The contents of the tags is repeated for each iteration. The `<xsl:value-of>` tag returns the value of an XPath. In this example, it used both to return the name parameter from each `<FIELD>` tag and to return the value of the `<FIELD>` tag itself.

```
[Variable: 'XSLT_String' = '<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="//ROOT">
    <TABLE>
      <TR>
        <xsl:for-each select="RECORD[1]/FIELD/@name">
          <TD><B><xsl:value-of select="self()" /></B></TD>
        </xsl:for-each>
      </TR>
      <xsl:for-each select="RECORD">
        <TR>
          <xsl:for-each select="FIELD">
            <TD><xsl:value-of select="self()" /></TD>
          </xsl:for-each>
        </TR>
      </xsl:for-each>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>']
```

The XML data is stored in XML_String.

```
[Variable: 'XML_String' = '<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <RECORD>
    <FIELD name="First Name">John</FIELD>
    <FIELD name="Last Name">Doe</FIELD>
  </RECORD>
</ROOT>']
```

The transformation is performed using the [XML_Transform] tag and the results are shown.

```
[XML_Transform: -XML=$XML_String, -XSL=$XSLT_String]
→ <TABLE>
  <TR>
    <TD><B>First_Name</B></TD>
    <TD><B>Last_Name</B></TD>
  </TR>
  <TR>
    <TD>John</TD>
    <TD>Doe</TD>
  </TR>
</TABLE>
```

XML Stream Data Type

The XML stream data type in Lasso automatically parses XML data which is stored in a variable. The member tags of the XML stream data type can then be used to inspect and change the XML data. *Table 7: XML Stream Data Type Tag* describes the tag which is used to convert string data to the XML stream data type.

The XML stream data type treats XML data as a stream of objects which will be consumed one by one until the end of the document is reached. This method of parsing XML documents is comparable to the SAX methodology.

The member tags of the XML data type can also be used to parse XML data. These methods are describes in the preceding *XML Data Type* section.

Table 7: XML Stream Data Type Tag

Tag	Description
[XMLStream]	Accepts a single parameter which is a string containing validly formatted XML data.

XML data from any source can be parsed and manipulated using Lasso by first storing the XML data in a variable and then casting it to the XML stream data type using the [XMLStream] tag. Lasso can work with XML data from a database field, XML file, remote Web application server, XML-RPC request, FTP site, etc. Or, Lasso can work with XML data that is created programmatically within a variable.

Navigating an XML Stream

An XML stream is made up of many objects called nodes. A node is a opening XML tag, a closing tag, an attribute of an XML tag, a string of text, CDATA, a processing instruction, a comment, or others. All of the available node types are detailed in *Table 8: XML Stream Node Types*.

Table 8: XML Stream Node Types

Type	Description
startElement	An opening XML tag. Opening XML tags are the only nodes that have attributes.
endElement	A closing XML tag.
attributes	An attribute of an opening XML tag.
text	The text contents of an XML tag.
cdata	The CDATA contents of an XML tag.
entityref	An entity reference. Often used for extended characters like & representing the ampersand &.
entitydecl	An entity declaration.
pi	A processing instruction. LassoScript embedded in an XML document would be considered a processing instruction. <? ... ?>
comment	A comment. These are formatted the same as HTML comments <!-- ... -->
document	The root of the XML document.
dtd	A document type declaration.
documentfrag	A fragment of a document.
notation	A notation.

The member tags which are used to set the current node of the XML stream are detailed in *Table 9: XML Stream Navigation Member Tags*. All of these tags return a boolean value if the desired operation could be performed. The current node for the stream is changed and the member tags in *Table 10: XML Stream Member Tags* can then be used to inspect the current node.

Table 9: XML Stream Navigation Member Tags

Tag	Description
[XMLStream->Next]	Advances to the next node of the XML stream. Returns True if successful or False if there are no more nodes.
[XMLStream->NextSibling]	Advances to the next sibling node, bypassing any child nodes. Returns True if successful or False if there are no more sibling nodes.
[XMLStream->MoveToAttribute]	Moves the position to the specified attribute. Requires a single parameter which is the attribute name to be moved to or an integer index to move to each attribute in order. Returns True if the current node could be changed.
[XMLStream->MoveToAttributeNamespace]	Moves the position to the specified attribute. Requires two parameters. The first parameter is the name of the attribute to be moved to. The second parameter is the URI of the namespace to be used. Returns True if the current node could be changed.
[XMLStream->MoveToFirstAttribute]	Moves the position to the first attribute associated with the current node. Returns True if successful or False if the current node has no attributes.
[XMLStream->MoveToNextAttribute]	Moves the position to the next attribute of the current node. Returns True if successful or False if there are no more attributes of the current node.
[XMLStream->MoveToElement]	Moves the position to the current node. Returns True if successful. This tag can be used to return to the node after moving to one or more attributes.

Navigation through an XML stream occurs only forward through the nodes. Each XML opening tag, closing tag, and other node types is visited in order using [XMLStream->Next]. The nodes are presented in the order they appear in the document without respect for the nesting of XML tags in the document.

```
<alpha name="value"> Some Text <beta> More Text </beta> </alpha>
```

For example, in the above XML document the following nodes will be visited in order: Starting at the document node. The `startElement` node representing the opening `<alpha>` tag. The text node `Some Text`. The `startElement` node representing the opening `<beta>` tag. The text node `More Text`. The `endElement` node representing the closing `</beta>` tag. And finally, the `endElement` node representing the closing `</alpha>` tag.

Node Attributes

As each node is visited its attributes can be fetched using one of the member tags detailed in *Table 10: XML Stream Member Tags*. These tags provide tools for inspecting the attributes and contents of a tag.

Table 10: XML Stream Member Tags

Tag	Description
[XMLStream->AttributeCount]	Returns the number of attributes of the current node.
[XMLStream->BaseURI]	Returns the base URI of the current node.
[XMLStream->Depth]	Returns the depth of the current node in the tree.
[XMLStream->GetAttribute]	Returns the value of an attribute of the current node. Requires one parameter which is the name of an attribute or an integer index to retrieve the attributes in order.
[XMLStream->GetAttributeNamespace]	Returns the value of an attribute of the current node. Requires two parameters. The first is the name of a parameter. The second is the URI for a namespace.
[XMLStream->HasAttributes]	Returns True if the current node has any attributes.
[XMLStream->HasValue]	Returns True if the current node can have a text value.
[XMLStream->isEmptyElement]	Returns True if the current node is empty.
[XMLStream->LocalName]	Returns the local name of the current node.
[XMLStream->LookupNamespace]	Returns the namespace for a prefix. Requires a single parameter which is the prefix to be looked up.
[XMLStream->NodeType]	Returns the type of the current node. The node types are identified in the following table.
[XMLStream->ReadAttributeValue]	Parses an attribute value into one or more Text and EntityReference nodes. Returns True if successful.
[XMLStream->ReadString]	Returns the text of the current node as a string.
[XMLStream->Name]	Returns the qualified name of the current node: "Prefix: LocalName".
[XMLStream->NamespaceURI]	Returns the URI defining the namespace associated with the current node.
[XMLStream->Prefix]	Returns the namespace prefix for the current node.
[XMLStream->Value]	Returns the text value of the current node if present.
[XMLStream->XMLLang]	Returns the xml:lang scope within which the current node resides.

XML Stream Example

The use of the XML stream tags depends largely on what type of XML document needs to be parsed. This example shows how a simple XML structure can be parsed and its attributes output to the browser.

This is the example XML data that will be processed for the example.

```
<xml>
  XML Data
  <tag param="value">
    A Tag
    <sub>A Sub-Tag</sub>
  </tag>
</xml>
```

To prepare to process the XML document it must be stored in a variable and then an [XMLStream] object is initialized.

```
[var: 'xml' = '<xml> ... </xml>']
[var: 'stream' = (xmlstream: $xml)]
```

The following code advances through the XML stream using [XMLStream->Next]. It prints out various attributes of the current node and then advanced through the node's attributes (if any).

```
<?LassoScript
while: $stream->next;
  output: $stream->nodetype + ': ' +
    "" + $stream->name + " = " + $stream->value + "";
  '<br>';
  if: ($stream->attributecount > 0) && ($stream->movetofirstattribute);
  var: 'more' = true;
  while: $more;
    output: loop_count + ' ' + $stream->nodetype + ': ' +
      "" + $stream->name + " = " + $stream->value + "";
    '<br>';
    var: 'more' = $stream->movetonextattribute;
  /while;
  $stream->(movetoelement);
/if;

/while;
?>
```

The results of running the code on the example XML document are shown below. Each node is output with its type, name, and value. The node for the opening <tag> has an attribute which is shown with a preceding numeral.

```

→ startElement: "xml" = ""
  text: "#text" = "XML Data"
  startElement: "tag" = ""
    1 attributes: "param" = "value"
    text: "#text" = "A Tag"
    startElement: "Sub" = ""
      text: "#text" = "A Sub-Tag"
    endElement: "sub" = ""
  endElement: "tag" = ""
endElement: "xml" = ""

```

This same code can be run on more complex XML documents to see how the XML stream tags report information about the different nodes. By adding actions when certain node types are encountered, this code can also be adapted into a tool that will parse XML and perform actions based on the contents.

XML-RPC

XML-RPC is a standard which allows remote procedure calls to be made between different servers on the Internet. A remote procedure call is similar to a CGI call (i.e. via [Include_URL]) to a different machine on the Internet, but by passing the parameters of the procedure call and results in a standard XML format, XML-RPC is more flexible than traditional CGIs.

One way to think of XML-RPC in Lasso is that it is a method of calling an LDML tag which happens to be located on a different Web server. Lasso can act as both ends of an XML-RPC call, enabling two Lasso servers to communicate with each other, or communication can be established between a Lasso server and another server that supports XML-RPC.

The first part of this section documents how to use the [XML_RPCCall] tag to make remote procedure calls. This technique is sufficient to make use of XML-RPC methods that are available on other servers. The second part of this section documents the low-level [XML_RPC] object and its methods for calling and responding to XML-RPC requests.

Note: The Extending Lasso 7 Guide contains additional information about how to create custom tags to respond to incoming remote procedure calls.

Calling a Remote Procedure

A remote procedure can be called using the [XML_RPCCall] tag. This tag uses the low-level XML-RPC data type to create a remote procedure call and to evaluate the results.

Table 11: [XML_RPCCall] Tag

Tag	Description
[XML_RPCCall]	Calls a remote procedure and returns the result. Accepts three parameters. -Host is the URL of the remote host. -Method is the method to be called. -Params is an array of parameters to be passed to the remote server.

The -Host parameter defaults to the current host. The -Method parameter is required, but defaults to Test.Echo for testing purposes. The -Params parameter is only required if the method requires parameters.

Errors are returned from the tag through the [Error_CurrentError]. This tag will report [Error_NoError] if no error occurred. Otherwise it will print out a detailed error message. The result of the [XML_RPCCall] tag when an error occurs is always Null.

To call a remote procedure:

Use the [XML_RPCCall] tag. In the following example the Test.Echo method on the current Lasso server is called. This method simply echoes its parameters back to the caller. The path to have Lasso process an incoming XML-RPC request is /Lasso/RPC.LassoApp.

```
[XML_RPCCall: -Host='http://127.0.0.1/Lasso/RPC.LassoApp',
  -Method='Test.Echo', -Params='Hello World!']
```

→ Hello World!

To list all available methods on a server:

Lasso supports a number of built-in XML-RPC methods. These are listed in *Table 12: XML-RPC Built-In Methods*. A list can be obtained directly from Lasso using the XML-RPC method System.ListMethods. Sample output is shown below.

```
[XML_RPCCall: -Host='http://127.0.0.1/Lasso/RPC.LassoApp',
  -Method='System.ListMethods']
```

→ (Array: (System.ListMethods), (System.MethodHelp), (System.MethodSignature), (System.MultiCall), (Test.Echo),

To call multiple methods on a server:

The System.MultiCall method can be used to call multiple methods on a remote server in a single request. This enables several XML-RPC methods to be called without the overhead of making individual HTTP connections to the remote server.

The following example performs two Test.Echo calls in a single System.MultiCall method.

```
[XML_RPCCall: -Host='http://127.0.0.1/Lasso/RPC.LassoApp',  
-Method='System.MultiCall', -Params=(Array:  
  (Map: 'MethodName'='Test.Echo', 'Params'='Hello World!'),  
  (Map: 'MethodName'='Test.Echo', 'Params'='Hello Again.'))]  
→ (Array: (Array: 'Hello World!'), (Array: 'Hello Again.'))
```

Note that the results are returned as an array with the return value of each particular method as an element.

Built-In Methods

Lasso supports a number of built-in XML-RPC methods which most XML-RPC processors are expected to have available. These built-in methods are implemented in `Startup.LassoApp` located in the `LassoStartup` folder.

Table 12: XML-RPC Built-In Methods

Method	Description
System.ListMethods	Returns an array of method names available on the server.
System.MethodHelp	Requires a method name as parameter. Returns a description of what the method does.
System.MethodSignature	Returns an error message since Lasso does not support message signatures.
System.MultiCall	Requires an array of maps each with a <code>MethodName</code> and a <code>Params</code> element. Returns an array of results for each of the individual methods.
Test.Echo	Echoes the parameters back to the caller.

Note: Lasso also defines a series of validator methods used to test the XML-RPC functionality for proper adherence to the standard.

XML-RPC and Built-In Data Types

Lasso automatically translates between XML-RPC data types and Lasso’s built-in data types. *Table 13: XML-RPC and Built-In Data Types* provides details about how data types are converted. Since Lasso performs two way

conversions XML-RPC calls to a Lasso server can be made without concern for data type conversions.

Table 13: XML-RPC and Built-In Data Types

XML-RPC Data Type	Lasso Equivalent
<i4> or <int>	Integer. XML-RPC supports only 32-bit signed integers.
<double>	Decimal. Double precision floating point number.
<boolean>	Boolean.
<dateTime.iso8601>	Date. Lasso automatically parses and formats XML-RPC date/times.
<string> or <base64>	String. Lasso stores both character and binary data in the string data type.
<struct>	Map. Individual <member> tags become elements of the map with <name> as the key and <value> as the value.
<array>	Array. Each <value> becomes an element of the array.

Note: Lasso supports 64-bit signed integers and greater floating point precision than many XML-RPC servers.

XML-RPC Data Type

Table 14: XML-RPC Data Type

Tag	Description
[XML_RPC]	Creates an XML_RPC object. Accepts an array of parameters for an outgoing XML_RPC call or for an incoming XML_RPC call that is to be processed.

Lasso supports calling a remote procedure through the [XML_RPC] data type. An instance of the [XML_RPC] data type is created with the parameters for the XML-RPC call, then the [XML_RPC->Call] tag is used to initiate the call and retrieve the results.

Table 15: [XML_RPC] Call Tag

Tag	Description
[XML_RPC->Call]	Calls a remote procedure. Requires two parameters. -URI is the location of the remote server. -Method is the name of the method to call on the remote server. The parameters of the request come from the XML_RPC object.

Note: The -URI parameter stands for Universal Resource Identifier.

To call a remote procedure using XML-RPC:

This example calls a remote procedure `GetPrice` which is available on a remote Lasso server at `http://rpc.example.com/RPC.LassoApp`. The remote procedure returns the price for a product based on name. The example has three steps: creating the `[XML_RPC]` object, calling the remote procedure, and interpreting the results.

- 1 Store the parameters for the remote procedure call in an `[XML_RPC]` parameter. The `GetPrice` procedure requires a single parameter which is the name of a product.

```
[Variable: 'MyRPC' = (XML_RPC: 'Widget')]
```

- 2 Call the `GetPrice` remote procedure on the desired server. The results are stored in a variable `MyResults`.

```
[Variable: MyResults= $MyRPC->(Call: -Method='GetPrice',  
-URI='http://rpc.example.com/RPC.LassoApp')]
```

- 3 Process the results. The first element of the returned array will be the price for the product. Finally, the price is displayed.

```
[Variable: 'Price' = $myResults->(Get: 1)]  
$[Variable: 'Price']
```

→ \$35.95

Processing an Incoming Remote Procedure Call

Lasso can processing incoming remote procedure calls in two ways.

- A custom tag can be created using the `[Define_Tag] ... [/Define_Tag]` tags with the `-RPC` parameter. The custom tag will be automatically made available through the `RPC.LassoApp`.
- Any Lasso format file can be used as the target for remote procedure calls. The methods of the `[XML_RPC]` data type can be used to interpret and process incoming calls.

The use of custom tags is the easiest way to process incoming remote procedure calls. Lasso handles the process of interpreting the method and parameters of each call and automatically returns the results to the caller. All XML-RPC calls are made to a single URL, i.e. `http://www.example.com/RPC.LassoApp`, making it easy to document what remote procedure calls the server supports.

Note: The creation of custom tags is covered in detail in *Chapter 3: Custom Tags* in the Extending Lasso 7 Guide.

To process remote procedure calls using a custom tag:

This example demonstrates how to create the `GetPrice` procedure used in the calling example. A custom tag named `[GetPrice]` will be created which accepts a single parameter, searches the `Products` table of a `Store` database, and returns the result. The `-RPC` parameter in the opening `[Define_Tag]` tag ensures that this procedure will be available through `RPC.LassoApp`.

```
[Define_Tag: 'GetPrice', -RPC, -Requires='Product']
  [Inline: -Search,
    -Database='Store',
    -Table='Products',
    'Product'=#Product]
  [Return: (Field: 'Price')]
[/Inline]
[/Define_Tag]
```

The tag can be called from a remote Lasso Professional 7 server using the `[XML-RPC]` tags. A call to the `GetPrice` remote procedure on the server at `http://rpc.example.com/` would look like as follows.

```
[Variable: MyResults= (XML_RPC: 'Widget')->(Call: -Method='GetPrice',
  -URI='http://rpc.example.com/RPC.LassoApp')]
[Variable: 'Price' = $myResults->(Get: 1)]
$[Variable: 'Price']
```

→ \$35.95

If more control is required beyond that provided by the built-in XML-RPC processing capabilities of Lasso then a custom format file can be created which processes incoming XML-RPC requests using the method of the `[XML_RPC]` data type directly.

An incoming XML-RPC request appears as a CGI call with POST parameters. An `[XML_RPC]` object should be initialized with the array of POST parameters from the `[Client_POSTArgs]` tag. The method and parameters of the incoming XML-RPC request can then be fetched with the member tags detailed in *Table 16: [XML_RPC] Processing Tags*.

Table 16: [XML_RPC] Processing Tags

Tag	Description
[XML_RPC->GetMethod]	Returns the method for an incoming XML_RPC request.
[XML_RPC->GetParams]	Returns an array of parameters for an incoming XML_RPC request.
[XML_RPC->Response]	Returns a response to an incoming [XML_RPC] request. Accepts two parameters. -Full is either True or False and determines whether full headers should be returned. -Fault is either True or False and determines whether an error response is returned.

To process an incoming XML-RPC request on a custom format file:

There are three steps to process an incoming XML-RPC request. First, the incoming request is parsed and the method and parameters are extracted. Second, the method and parameters are processed. Finally, the results are formatted and returned to the caller.

- 1 The incoming XML-RPC request is processed by passing [Client_POSTArgs] to the [XML_RPC] tag. The method and parameters of the incoming request are then extracted with the [XML_RPC->GetMethod] and [XML_RPC->GetParams] tags.

```
[Variable: 'myRPC' = (XML_RPC: (Client_POSTArgs))]  
[Variable: 'myMethod' = $myRPC->GetMethod]  
[Variable: 'myParameters' = $myRPC->GetParameters]
```

- 2 Since a single format file might process many different XML-RPC methods [Select] ... [Case] ... [/Select] tags are used to determine what code to process.

```
[Select: $myMethod]  
[Case: 'GetPrice'  
[Inline: -Search,  
-Database='Store',  
-Table='Products',  
'Product'=$MyParameters]  
[Variable: 'Response'=(Field: 'Price')]  
[/Inline]  
[/Select]
```

- 3 The response is sent back to the caller of the remote procedure by outputting the result of the [XML_RPC->Response] tag with the results of the remote procedure. -Full is set to True so full HTTP headers will be

returned to the caller and -Fault is set to False indicating that the XML-RPC call was successful.

```
[Variable: 'myRPC' = (XML_RPC: $Result)]
[ $myRPC->(Response: -Full=True, -Fault=False)]
```

SOAP

The Simple Object Access Protocol (SOAP) is a standard which allows remote procedure calls to be made between different servers on the Internet. A remote procedure call is similar to a CGI call (i.e. via [Include_URL]) to a different machine on the Internet, but by passing the parameters of the procedure call and results in a standard XML format, SOAP is more flexible than traditional CGIs.

Lasso provides built-in support for incoming SOAP requests by allowing properly defined custom tags to respond to these requests automatically. Lasso does not provide built-in support for calling remote SOAP services, but the XML tools in this chapter and [Include_URL] can be used to easily handcraft formatted XML for a SOAP request and to parse the return value.

The first part of this section documents how to use the [Include_URL] tag to make remote procedure calls using SOAP and how to parse the results. The second part of this section documents how to create custom tags that automatically respond to incoming SOAP requests and how to create format files that respond to incoming SOAP requests using either GET or POST methodology.

Note: The Extending Lasso 7 Guide contains additional information about how to create tags to respond to incoming remote procedure calls.

Calling a Remote SOAP Procedure

A remote procedure can be called using the [Include_URL] tag with an appropriately formatted XML SOAP envelope. The SOAP envelope should be included in the documentation of the SOAP service and can be easily modified to include different parameters using Lasso's string tools.

To call a remote SOAP procedure:

This example calls a spell checking procedure that is available as a SOAP service. The input to the SOAP service are the words to be spell checked and the response includes one or more suggestions for proper spellings.

- 1 Format the SOAP envelope according to the documentation for the service. There is a lot of XML in the envelope, but only one part needs to

be customized in order to craft different spell check requests. The words `bleu wrld` in the `<phrase> ... </phrase>` tags will be spell checked. Additional spell check requests can be created by changing these words and no other parts of the envelope.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/1999/XMLSchema'>
  <SOAP-ENV:Body>
    <ns1:doSpellingSuggestion
      xmlns:ns1='urn:GoogleSearch'
      SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
      <phrase xsi:type='xsd:string'>bleu wrld</phrase>
    </ns1:doSpellingSuggestion>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- 2 Store the envelope in a variable. Here the envelope is simply placed in a variable. Different SOAP requests can be constructed using search/replace on a template envelope or by appending the start of the envelope, the phrase to be checked, and the end of the envelope together.

```
[Variable: 'SOAP_Envelope' = '<SOAP-ENV:Envelope ... ']
```

- 3 Use `[Include_URL]` to send the SOAP envelope to the appropriate URL. The SOAP envelope must be sent as a POST like form parameters using the `-PostParams` parameter of `[Include_URL]`. The content-type of the request must be set to `text/xml` using the `-ExtraMIMEHeaders` parameter. The result will be stored in a variable.

```
[Variable: 'Result' = (Include_URL: 'http://soap.example.com/SpellCheck',
  -PostParams=$SOAP_Envelope,
  -ExtraMIMEHeaders=(Array: 'content-type'='text/xml'))]
```

- 4 The result will either include a fault code or proper response. First, check to see if it is a fault code and display an appropriate error message. The code below uses the `[XML->Extract]` tag with an XPath of `//faultcode` and `//faultstring` to extract these XML entities if they exist. The `[Protect] ... [/Protect]` tags will ensure that if there is no fault code the error is suppressed.

```
[Variable: 'XMLResult' = (XML: $Result)]
[Protect]
[Variable: 'FaultCode' = $XMLResult->(Extract: '//faultcode')->(Get:1)->Contents]
[Variable: 'FaultString' = $XMLResult->(Extract: '//faultstring')->(Get:1)->Contents]
[/Protect]
[Output: $FaultString + ' (' + $FaultCode ')']
```

If an error occurs this will result in an error message like the following. This error indicates that one of the namespace entries in the SOAP envelope is missing.

→ Unable to determine object id from call: is the method element namespaced?
(SOAP-ENV:Server.BadTargetObjectURI)

- 5 Finally, the results of a successful operation are formatted for output. The results are extracted from the <return> tag using the [XML->Extract] tag with an XPath of //return. The resulting array is looped through to pull the contents out of each return tag and create the final output for the client.

```
[Variable: 'XMLResult' = (XML: $Result)]
[Variable: 'Output' = ""]
[Protect]
  [Iterate: $XMLResult->(Extract: '//return'), (Var: 'temp')]
    [$Output += $Temp->Contents + ' ']
  [/Iterate]
[/Protect]
[Output: 'Suggestions: ' + $Output]
```

The results of a successful SOAP call are displayed below. The suggested spelling is correct.

→ Suggestions: blue world

Note: This example is based on the Google spell checking service, but has been simplified to provide a more concise example.

Processing an Incoming SOAP Call

Lasso can processing incoming SOAP remote procedure calls in two ways.

- A custom tag can be created using the [Define_Tag] ... [/Define_Tag] tags with the -SOAP parameter. The custom tag will be automatically made available through the RPC.LassoApp. All parameters of the tag must have explicit types defined using -Type parameters and the return type of the tag must be defined using the -ReturnType parameter.
- Any Lasso format file can be used as the target for remote procedure calls through the GET method. The URL parameters are interpreted normally and a SOAP envelope is returned as a result.

The use of custom tags is the easiest way to process incoming SOAP remote procedure calls. Lasso handles the process of interpreting the method and parameters of each call and automatically returns the results to the caller. All SOAP calls are made to a single URL, i.e. <http://www.example.com/RPC.LassoApp>, making it easy to document what remote procedure calls the server supports.

Note: The creation of custom tags is covered in detail in *Chapter 3: Custom Tags* in the Extending Lasso 7 Guide.

To process SOAP remote procedure calls using a custom tag:

This example demonstrates how to create a `GetPrice` procedure. A custom tag named `[GetPrice]` will be created which accepts a single parameter, searches the `Products` table of a `Store` database, and returns the result.

The `-SOAP` parameter in the opening `[Define_Tag]` tag ensures that this procedure will be available through `RPC.LassoApp` as a SOAP procedure. The `-Type` parameter is required to specify the type of the `Product` parameter. The `-ReturnType` parameter specifies what type the return value of the tag will be.

```
[Define_Tag: 'GetPrice', -SOAP,
  -Requires='Product', -Type='String',
  -ReturnType='String']
  [Inline: -Search,
    -Database='Store',
    -Table='Products',
    'Product'='##Product']
  [Return: (Field: 'Price')]
[/Inline]
[/Define_Tag]
```

The tag can be called from a remote Lasso Professional 7 server using the techniques documented earlier.

To process an incoming SOAP request in a custom format file:

If more control is required beyond that provided by the built-in XML-RPC processing capabilities of Lasso then a custom format file can be created which processes incoming SOAP requests directly. An incoming SOAP request appears either as a GET request with URL parameters or as a CGI call with a SOAP envelope in the POST parameters.

- SOAP procedures can be called using GET parameters in a URL. These parameters can be processed using `[Action_Param]` just like any URL parameters. The response to the SOAP procedure should be a properly formatted SOAP envelope or SOAP fault code with a content type of `text/xml`.

In the following code the `GetPrice` method is implemented for the following URL. In this example the price for a `Widget` will be returned.

```
http://www.example.com/SOAP/GetPrice.Lasso?Product=Widget
```

The code of the page performs the database search and returns a SOAP envelope with a `<return>` tag if the result is good or fault tags if the result is undefined.

```

[Content_Type: 'text/xml']
[Inline: -Search,
  -Database='Store',
  -Table='Products',
  'Product'=(Action_Param: 'Product')]
[Variable: 'Response'=(Field: 'Price')]
[/Inline]
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
[If: $Response != ""]
  <ns1:GetPrice
    xmlns:ns1="http://www.example.com/SOAP/GetPrice.Lasso"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <return xsi:type="xsd:string">[Variable: 'Response']</return>
  </ns1:doSpellingSuggestionResponse>
[Else]
  <SOAP-ENV:Fault>
    <faultcode>ERROR</faultcode>
    <faultstring>No Price Found</faultstring>
    <faultactor>[Response_FilePath]</faultactor>
  </SOAP-ENV:Fault>
[/If]
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

- SOAP procedures can be called by embedding a SOAP envelope as the POST parameter in a Web request. The raw POST parameter can be fetched using [Client_PostArgs]. This returns a string that can be fed into the [XML] tag for further processing.

In the following example this SOAP envelope will be considered a valid request. The product is contained in <product> tags and in this example the price for a Widget should be returned.

```

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/1999/XMLSchema'>
<SOAP-ENV:Body>
  <ns1:GetPrice
    xmlns:ns1s='http://www.example.com/SOAP/GetPrice.Lasso'
    SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>

```



```

        <product xsi:type='xsd:string'>Widget</product>
    </ns1:GetPrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The incoming request is parsed using the following code. At the end, the variable `Product` contains the name of the product whose price should be returned. The `[Protect] ... [/Protect]` tags ensure that the absence of `<product>` tags does not cause a syntax error.

```

[Variable: 'SOAPEnvelope' = (Client_PostArgs)]
[Variable: 'XMLEnvelope' = (XML: $SOAPEnvelope)]
[Variable: 'Price' = ""]
[Protect]
    [Variable: 'Price' = $XMLEnvelope->(Extract: '//product')->(Get:1)->Contents]
[/Protect]

```

The rest of the page performs the database search and returns a SOAP envelope with a `<return>` tag if the result is good or fault tags if the result is undefined.

```

[Content_Type: 'text/xml']
[Inline: -Search,
    -Database='Store',
    -Table='Products',
    'Product'=(Variable: 'Product')]
[Variable: 'Response'=(Field: 'Price')]
[/Inline]
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
    [If: $Response != ""]
        <ns1:GetPrice
            xmlns:ns1="http://www.example.com/SOAP/GetPrice.Lasso"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <return xsi:type="xsd:string">[Variable: 'Response']</return>
        </ns1:doSpellingSuggestionResponse>
    [Else]
        <SOAP-ENV:Fault>
            <faultcode>ERROR</faultcode>
            <faultstring>No Price Found</faultstring>
            <faultactor>[Response_FilePath]</faultactor>
        </SOAP-ENV:Fault>
    [/If]
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Serving XML

XML data which is created within a variable, stored in a database, or read from a file on the Web serving machine can be served in place of the current format file using the [XML_Serve] tag. When this tag is called processing of the current format file is aborted and the specified XML data is served to the site visitor.

The visitor's Web browser will determine how the XML data is formatted. Many Web browsers will show XML data in outline form where the individual tags can be collapsed or expanded to view different portions of the data.

Table 17: [XML_Serve] Serving Tags

Tag	Description
[XML_Serve]	Returns XML data in place of the current format file. The first parameter is the XML data to be served. Optional -File parameter allows the name of the XML data to be specified. Optional -Type parameter allows the MIME type to be overridden from the default of text/xml.

To serve XML data:

Use the [XML_Serve] tag. The following example serves some simple XML data in place of the current format file. No tags after the [XML_Serve] tag will be processed.

```
[Variable: 'XMLData' = '<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <ROW>
    This is XML data.
  </ROW>
</ROOT>']
[XML_Serve: $XMLData]
```

Formatting XML

XML data should be served using the MIME type of text/xml and a UTF-8 character set. The [Content_Type] tag can be used to set the MIME type and character set of a page served by Lasso. This tag simply adjusts the header of the page served by Lasso, it does not perform any conversion of the data on the page.

To specify that a format file contains XML:

Use the following tag as the very first line of any files which contain XML data. Notice that the tag accepts only a single parameter, the `charset` argument is appended to the `MIME` type argument with a semi-colon ;.

```
[Content_Type: 'text/xml; charset=UTF-8']
```

To format XML:

Most XML pages have the following format, an `<?XML ... ?>` declaration followed by a root tag that surrounds the entire contents of the file. This is similar to the `<html>` tag that typically surrounds an entire HTML page. The following example shows a `<ROOT> ... </ROOT>` tag with a single `<ROW> ... </ROW>` tag inside.

```
[Content_Type: 'text/xml; charset=UTF-8']
<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
  <ROW>
    This is XML data.
  </ROW>
</ROOT>
```

To encode data within XML:

The data within XML tags and tag parameters should be XML encoded. The `[Encode_Set] ... [/Encode_Set]` tags can be used to change the default encoding for all substitution tags in an entire XML page. The following example shows an XML page with an enclosing set of `[Encode_Set] ... [/Encode_Set]` tags. The value of the `[Variable]` tag will be XML encoded, ensuring that it is recognized properly by an XML parser.

```
[Content_Type: 'text/xml; charset=UTF-8']
<?xml version="1.0" encoding="UTF-8" ?>
[Encode_Set: -EncodeXML]
<ROOT>
  <ROW>
    [Variable: 'XML_Data']
  </ROW>
</ROOT>
[/Encode_Set]
```

Tags which return XML tags should not have their values encoded. Tags which return XML data require an `-EncodeNone` encoding keyword in order to ensure that the angle brackets and other markup characters are not encoded into XML entities. The following example shows a variable that returns an entire `<ROW> ... </ROW>` tag. The `[Variable]` tag has an `-EncodeNone` keyword so the angle brackets within the XML data are not encoded.

```

[Content_Type: 'text/xml; charset=UTF-8']
[Variable: 'XML_Data' = '<ROW><p>This is XML data.<ROW>']
<?xml version="1.0" encoding="UTF-8" ?>
[Encode_Set: -EncodeXML]
  <wml>
    [Variable: 'XML_Data', -EncodeNone]
  </wml>
[/Encode_Set]

```

XML Templates

Lasso includes a collection of XML templates that you can incorporate into your own Web site or customize to use a different DTD or schema. In order to use the templates, you must construct a Lasso action which uses the XML template as its response.

The templates are contained in Documentation/4-LanguageGuide/Examples/XML/ folder within the Lasso Professional 7 application folder. In order to use these examples, the entire XML folder should be copied into the Web server root. The examples in this section assume the XML folder can be reached at the root of the Web server by the following URL.

<http://www.example.com/XML/>

- **FileMaker Pro** templates allow data to be published using the same formats as those provided with FileMaker Pro Data Source Object (DSO) and FileMaker Pro (FileMaker) templates are provided including versions with DTDs and schemas. Each of the templates is described in *Table 18: FileMaker Pro XML Templates*. All of the templates are included in the folder FileMaker within the XML folder.
- **SQL Server** templates allow data to be published using some of the formats provided with Microsoft SQL Server. Templates are provided for Raw SQL results and for results structured as Auto Elements (Elem). Each of the templates includes versions with DTDs and schemas. They are described in *Table 19: SQL Server XML Templates*. All of the templates are included in the folder SQLServer within the XML folder.

Each template can be used as the response to a Lasso action that returns records. The templates are written in a database-independent fashion and build their DTD or schema based on the actual field names which define the results that they are formatting.

Table 18: FileMaker Pro XML Templates

Template	Description
dso_xml.lasso	Data Source Object template uses the name of each field in the database as the name of an XML tag. Root tag is <FMPDSORESULT> which contains <ROW> tags that contain individual field tags.
dso_xml_dtd.lasso	Includes a dynamically generated DTD.
dso_xml_schema.lasso	Includes a dynamically generated Schema.
fmp_layout_xml.lasso	FileMaker Pro Layout template includes <LAYOUT>, <FIELD>, and <VALUelist> tags which describe a FileMaker Pro layout. Root tag is <FMPXMLLAYOUT> which contains <ERRORCODE> <PRODUCT> and <LAYOUT> tags.
fmp_layout_xml_dtd.lasso	Includes a dynamically generated DTD.
fmp_layout_xml_schema.lasso	Includes a dynamically generated Schema.
fmp_xml.lasso	FileMaker Pro results template includes database structure and <RESULTS> tag with <ROW> and <COL> sub-tags. Root tag is <FMPXMLRESULT> which contains <ERRORCODE> <PRODUCT> <DATABASE>, <METADATA> and <RESULTS> tags.
fmp_xml_dtd.lasso	Includes a dynamically generated DTD.
fmp_xml_schema.lasso	Includes a dynamically generated Schema.

Table 19: SQL Server XML Templates

Template	Description
sql_xml_raw.lasso	Includes each record in a single <ROW> tag. Root tag is <ROOT> which contains <ROW> tags. Each field is specified as a parameter of the <ROW> tags.
sql_xml_raw_dtd.lasso	Includes a dynamically generated DTD.
sql_xml_raw_schema.lasso	Includes a dynamically generated Schema.
sql_xml_elem.lasso	Includes each record in a single <ROW> tag. Root tag is <ROOT> which contains <ROW> tags. Each field is specified as a tag named the same as the field name within the <ROW> tags.
sql_xml_elem_dtd.lasso	Includes a dynamically generated DTD.
sql_xml_elem_schema.lasso	Includes a dynamically generated Schema.

To use a template with an [Inline] ... [/Inline] action:

Specify a search within [Inline] ... [/Inline] tags and use an [Include] tag to insert the desired template to format the results. A [Content_Type] tag is required

at the top of the file containing the [Inline] ... [/Inline] tags so the MIME type of the returned data will be properly specified. The following example finds all records in a Contacts database and formats the results using the FileMaker DSO template stored at /XML/FileMaker/dso_xml.lasso.

```
[Content_Type: 'text/xml; charset=UTF-8']
[Inline: -Database='Contacts', -Table='People', -KeyField='ID', -FindAll]
  [Include: '/XML/FileMaker/dso_xml.lasso']
[/Inline]
```

To use a template with an HTML form-based action:

Specify a search within an HTML form. The -Response to the form should be a template file. The following example finds all records in a Contacts database and formats the results using the FileMaker DSO template stored at /XML/FileMaker/dso_xml.lasso.

```
<form action="/Action.Lasso" method="POST">
  <input type="hidden" name="-Database" value="Contacts">
  <input type="hidden" name="-Table" value="People">
  <input type="hidden" name="-KeyField" value="ID">
  <input type="hidden" name="-Response"
    value="/XML/FileMaker/dso_xml.lasso">
  <input type="submit" name="-FindAll" value="Find All Contacts">
</form>
```

To use a template with a URL-based action:

Specify a search within a URL. The -Response specified in the URL should be a reference to a template file. The following example finds all records in a Contacts database and formats the results using the FileMaker DSO template stored at /XML/FileMaker/dso_xml.lasso.

```
<a href="/Action.lasso?-Datasource=Contacts&
  -Table=People&
  -KeyField=ID&
  -Response=/XML/FileMaker/dso_xml.lasso&
  -FindAll">
  Find All Contacts
</a>
```

How to Customize

The XML templates are good examples of data source-independent design and can be used as the starting point of an automatic publishing system based on XML.

The templates are also a good starting point to create XML format files that are industry specific or to a particular database structure. Starting with a

DTD or schema, an XML format file can be created that outputs data from a data source in precisely the format required.

Custom XML templates will need to be created in order to take advantage of relationships, repeating fields, stored images, or portals specific to any given data source.

30

Chapter 30

Portable Document Format

This chapter describes how to create files in Portable Document Format (PDF) using LDML 7.

- *Overview* introduces PDF support in Lasso Professional 7.
- *Creating PDF Documents* describes how to create PDF documents using LDML tags, and how to use existing PDF documents as templates.
- *Creating Text Content* describes how to add text to a PDF variable using LDML tags.
- *Creating and Using Forms* describes how to add forms to a PDF variable using LDML tags, and also discusses how PDF forms can be used to submit data to a database using Lasso.
- *Creating Tables* describes how to create and insert tables in a PDF variable using LDML tags.
- *Creating Graphics* describes how to create and insert graphics in a PDF variable using LDML tags.
- *Creating Barcodes* describes how to create and insert barcodes in a PDF variable using LDML tags.
- *Example PDF Files* provides complete examples of using LDML to create PDF files with text, forms, tables, graphics, and barcodes.
- *Serving PDF Files* describes how to display a PDF file within the context of a Lasso format file.

Overview

Lasso Professional 7 provides support for Portable Document Format (PDF) files, allowing PDF documents to be created using LDML. The PDF file format is a widely-accepted standard for electronic documentation, and facilitates superb printer-quality documents from simple graphs to complex forms such as tax forms, escrow documents, loan applications, stock reports, and user manuals. For more information on PDF technology, see the following URL.

<http://www.adobe.com/products/acrobat/adobe.pdf.html>

Implementation Note: The [PDF_...] tags in LDML 7 are implemented in LJAPI, and based on the iText Java library. For more information on the iText Java library, visit <http://www.lowagie.com/iText>.

Introduction to Creating PDF Files

PDF files are created in LDML by setting a variable as a [PDF_Doc] object, and using various member tags and other [PDF_...] tags to add data to the variable. The PDF is then written to file when the format file containing all code is served by the Web server.

To create a basic PDF file using LDML:

The following shows an example of creating and outputting a PDF file named MyFile.pdf using the [PDF_...] tags.

```
[Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',
                    -Format='PDF',
                    -Size='A4',
                    -Margin=(Array: 144.0, 144.0, 72.0, 72.0))]
[Var: 'Font'=(PDF_Font: -Face='Helvetica', -Size=36)]
[Var: 'Text'=(PDF_Text:'I am a PDF document', -Font=$Font)]
[$MyFile->(Add: $Text)]
[$MyFile->Close]
```

In the example above, a variable named MyFile is set to a [PDF_Doc] type for a file named MyFile.pdf. A single font type is defined for the document using the [PDF_Font] tag. Then, the text I am a PDF document is defined using the [PDF_Text] tag, and added using the [PDF_Doc->Add] member tag. The PDF is then written to file upon execution of the [\$MyFile->Close] tag.

This chapter explains in detail how these and other tags are used to create and edit PDF files. This chapter also shows how to output a PDF file to a client browser within the context of a format file, which is described in the *Serving PDF Files* section of this chapter.

File Permissions

This section describes the file permission requirements for creating PDF files on a Web server using LDML 7. In order to successfully create PDF files, the following conditions must be met.

- When creating PDF files using the [PDF_...] tags, the current user must have Create Files, Read Files, and Write Files permissions allowed in the *Setup > Security > Files* section of Lasso Administration, and the folder in which the PDF will be created must be available to the user within the Allow Path field.
- Any file extensions being used by the [PDF_...] tags must be allowed in the *Setup > Global Settings > Settings* section of Lasso Administration. This can include .pdf, .jpg, and .gif.
- When creating files, Lasso Service must be allowed to write to the folder by the operating system (i.e. the Lasso system user in Mac OS X and Linux). For more information, see *Chapter 20: Files and Logging*.

Creating PDF Documents

PDF documents are initialized and created using the [PDF_Doc] tag. This is the basic tag used to create PDF documents with Lasso, and is used in concert with all tags described in this chapter.

Table 1: [PDF_Doc] Tag and Parameters

Tag	Description
[PDF_Doc]	Initializes a PDF document. Uses optional parameters which set the basic specifications of the file to be created. Data is added to the variable using [PDF_Doc] member tags, which are described throughout this chapter.
-File	Defines the file name and path of the PDF document. If omitted, the PDF document is created in RAM (see the <i>Serving PDF Files</i> section of this chapter for more information). If a file name is specified without a folder path, the file is created in the same location as the format file containing the [PDF_...] tags.
-Size	Define the page size of the document. Values for this parameter are standard print sizes, and can be A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, B0, B1, B2, B3, B4, B5, ARCH_A, ARCH_B, ARCH_C, ARCH_D, ARCH_E, FLSA, FLSE, HALFLETTER, LEDGER, LEGAL, LETTER, NOTE, and TABLOID. Defaults to A4 if not used. Optional.

-Height	Defines a custom page height for the document. Accepts a decimal value which represents the size in points. This can be used with the -Width parameter instead of the -Size parameter. Optional.
-Width	Defines a custom page width for the document. Requires a decimal value which represents the size in points. This can be used with the -Height parameter instead of the -Size parameter. Optional.
-Margins	Defines the margin size for the page. Requires an array of four decimal values, which define the left, right, top, and bottom margins for the page (Left, Right, Top, Bottom). Optional.
-Color	Defines the initial text color of the PDF document. Requires a hex color string. Defaults to '#000000' if not used. Optional.
-UseDate	Adds the current date and time to the file header. Optional.
-NoCompress	Produces a PDF without compression to allow PDF code to be viewed. PDF files are compressed by default if not used. Optional.
-PageNo	Sets the starting page number for the PDF document. Requires an integer value, which is the page number of the first page. Optional.
-PageHeader	Sets text that will be displayed at the top of each page in the PDF. Requires a text string as a value. Optional.
'Header'='Content'	Adds defined file headers to the PDF document. 'Header' is replaced with the name of the file header (e.g. Title, Author), and 'Content' is replaced with the header value. Optional.

The examples below show creating basic PDF files, however these files contain little or no data. Various types of data can be added to these files using the tags described in the remainder of this chapter.

To start a basic PDF file:

Use the [PDF_Doc] tag to create a PDF file to a hard drive location on the Web server. Use the -File parameter to define the location and file name, and the -Size parameter to define a pre-defined standard size. This basic example creates a blank, one-page PDF document.

```
[Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf', ,
                        -Size='A4')]
```

To start a PDF file with a custom page size:

Use the [PDF_Doc] tag with the -Height and -Width parameters to define a custom page size in points. One inch is equal to 72 points.

```
[Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',
                        -Height='648.0',
                        -Width='468.0')]
```

To start a PDF file with custom margins:

Use the [PDF_Doc] tag with the -Margin parameter to define a custom page size (in points). The following example adds a margin of 72 points (one inch) to the left and right sides of the page, but adds no margin to the top and bottom. This example also adds the date and time of creation to the file header using the -UseDate parameter.

```
[Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',
                        -Size='A4',
                        -Margin=(Array: 72.0, 72.0, 0.0, 0.0),
                        -UseDate)]
```

To start an uncompressed PDF file:

Use the [PDF_Doc] tag with the -NoCompress parameter.

```
[Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',
                        -Size='A4',
                        -NoCompress)]
```

To start a PDF file with custom file headers:

Use the [PDF_Doc] tag with appropriate 'Header'='Content' parameters.

```
[Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',
                        -Size='A4',
                        'Title'='My PDF File',
                        'Subject'='How to create PDF files',
                        'Author'='John Doe')]
```

Adding Content to PDFs

In LDML 7, there are several different types of data that can be added to a PDF document. Many of these types are first defined as objects using tags such as [PDF_Text], [PDF_List], [PDF_Image], [PDF_Table], or [PDF_BarCode], and then added to a [PDF_Doc] variable using the [PDF_Doc->Add] member tag. Each data type object is described separately in subsequent sections of this chapter.

Table 2: [PDF_Doc->Add] Tag and Parameters

Tag	Description
[PDF_Doc->Add]	Adds a PDF content object to a document. This can be used to add [PDF_Text], [PDF_List], [PDF_Image], [PDF_Table], or [PDF_BarCode] objects. If no position information is specified then the object is added to the flow of the page, otherwise it is drawn at the specified location. Requires one parameter, which is the object to be added. Optional parameters are described below.
-Align	Sets the alignment of the object in the page ('Left', 'Center', or 'Right'). Defaults to 'Left'. Works only for [PDF_Image] and [PDF_BarCode] objects. Optional.
-Wrap	Keyword parameter specifies that text should flow around the embedded object. Works only for [PDF_Image] and [PDF_BarCode] objects. Optional.
-Left	Specifies the placement of the object relative to the left side of the document. Requires a decimal value, which is the placement offset in points. Works only for [PDF_Image] and [PDF_BarCode] objects. Optional.
-Top	Specifies the placement of the object relative to the top of the document. Requires a decimal value, which is the placement offset in points. Works only for [PDF_Image] and [PDF_BarCode] objects. Optional.
-Height	Scales the object to the specified height. Requires a decimal value which is the desired object height in points. Works only for [PDF_Image] and [PDF_BarCode] objects. Optional.
-Width	Scales the object to the specified width. Requires a decimal value which is the desired object width in points. Works only for [PDF_Image] and [PDF_BarCode] objects. Optional.

For examples of using the [PDF_Doc->Add] tag to add text, image, table, and barcode PDF objects to a [PDF_Doc] variable, see the corresponding sections in this chapter.

Adding Pages

If the content of a PDF document will span more than one page, additional pages can be added using special [PDF_Doc] member tags. These tags signal where pages start and stop within the flow of the LDML PDF creation tags.

Table 3: PDF Page Tags

Tag	Description
[PDF_Doc->AddPage]	Adds additional blank pages to the [PDF_Doc] variable. When used, this tag ends in the current page and starts a new page.
[PDF_Doc->AddChapter]	Adds a page with a named chapter title (and bookmark) to a [PDF_Doc] variable. Requires a text string or [PDF_Text] object as a parameter, which specifies the chapter title. An additional -Number parameter sets an integer chapter number for the chapter. An optional -HideNumber parameter specifies that no number will be shown.
[PDF_Doc->SetPageNumber]	Sets a page number for a new page. Requires an integer value.
[PDF_Doc->GetPageNumber]	Returns the current page number.

To start a new page:

Use the [PDF_Doc->AddPage] tag. The following example ends a preceding page, and starts a new page.

```
[$MyFile->(Add:'Thus, ends the discussion on page 1.')]
[$MyFile->AddPage]
[$MyFile->(Add:'On page 2, we will discuss something else.')]

```

To add a chapter title:

Use the [PDF_Doc->AddChapter] tag. The following example adds a page with the text 30. Important Chapter to the [PDF_Doc] variable with a defined chapter number of 30.

```
[$MyFile->(AddChapter:'Important Chapter', -Number=30)]

```

To set the page number for a page:

Use the [PDF_Doc->SetPageNumber] tag. The following example sets a page number of 5 for the current page.

```
[$MyFile->(SetPageNumber: 5)]

```

To return the current page number:

Use the [PDF_Doc->GetPageNumber] tag. The following example returns a page number of 1 when used within the first page of the document.

```
[$MyFile->GetPageNumber] → 1

```

Adding Pages from Existing PDFs

Pages in existing PDF documents can be added to a [PDF_Doc] variable using the [PDF_Read] tag. This tag makes it possible to use existing PDF documents as templates.

Note: Lasso cannot change existing text or graphics that are contained within a PDF document read in using [PDF_Read]. Instead, Lasso is able to overlay text, graphics, and other elements on the PDF.

Table 4: PDF Read Tags

Tag	Description
[PDF_Read]	Casts an existing PDF document on the server as an LDML object. This object can be added to a [PDF_Doc] variable using the [PDF_Doc->InsertPage] tag, and then modified. Requires a -File parameter, which specifies the name and path to a PDF file on the server to read.
[PDF_Read->PageCount]	Returns the integer number of pages in the document.
[PDF_Read->PageSize]	Returns the dimensions of the first page in the PDF as an array of width and height point values (Array: Width, Height). An optional integer parameter specifies the page number in the document to return the size of.

To read in an existing PDF document:

In order to work with an existing PDF document, it must first be cast as a Lasso variable using the [PDF_Read] tag.

```
[Var:'Old_PDF'=(PDF_Read:-File='/documents/somepdf.pdf')]
```

To determine the attributes of an existing PDF document:

The number of pages and the dimensions of an existing PDF document can be returned using the [PDF_Read->PageCount] and [PDF_Read->PageSize] tags on a defined [PDF_Read] variable.

```
[Var:'Old_PDF'=(PDF_Read:-File='/documents/somepdf.pdf')]
Number of pages: [$Old_PDF->PageCount]<br>
Page size: [$Old_PDF->(PageSize: 1)]
```

Once an existing PDF document has been cast as a Lasso object using [PDF_Read], it may be added to a [PDF_Doc] variable using the [PDF-Doc->InsertPage] tag.

Table 5: Page Insertion Tag and Parameters

Tag	Description
[PDF_Doc->InsertPage]	Inserts a page from a [PDF_Read] object into a [PDF_Doc] variable. Requires the name of a [PDF_Read] variable, followed by a comma and the number of the page to insert. This tag has many optional parameters for specifying how an existing page should be inserted into a [PDF_Doc] variable. These parameters are explained below.
-NewPage	Keyword parameter specifying that the new page should be appended at the end of the document. Otherwise the page is drawn over the first page in the [PDF_Doc] variable by default.
-Top	If the page being inserted is shorter than the current pages in the [PDF_Doc] variable, this parameter may be used to specify the offset of the new page from the top of the current page frame in points.
-Left	If the page being inserted is not as wide the current pages in the [PDF_Doc] variable, this parameter may be used to specify the offset of the new page from the left of the current page frame in points.
-Width	Scales the inserted page by width. Requires either a point width value, or a percentage string (e.g. 50%).
-Height	Scales the inserted page by height. Requires either a point height value, or a percentage string (e.g. 50%).

To insert an existing page into a new PDF document:

Use the [PDF_Doc->InsertPage] tag with a defined [PDF_Read] variable. The example below makes the first page of the somepdf.pdf PDF the first page of the [PDF_Doc] variable. Content may then be overlaid on top of the new page using the tags described in the rest of this chapter.

```
[Var:'New_PDF'=(PDF_Doc: -File='MyFile.pdf',
                    -Format='PDF',
                    -Size='A4')]
[Var:'Old_PDF'=(PDF_Read:-File=/documents/somepdf.pdf)]
[$New_PDF->(InsertPage: $Old_PDF, 1)]
```

To insert an existing page at the end of a new PDF document:

Use the [PDF_Doc->InsertPage] tag with the optional -NewPage parameter. The example below adds the first page of the somepdf.pdf PDF after all existing pages in the [PDF_Doc] variable.

```
[Var:'New_PDF'=(PDF_Doc: -File='MyFile.pdf',  
                    -Format='PDF',  
                    -Size='A4'))]  
[Var:'Old_PDF'=(PDF_Read:-File='/documents/somepdf.pdf')]  
[$New_PDF->(InsertPage: $Old_PDF, 1, -NewPage)]
```

To place an inserted page:

Use the [PDF_Doc->InsertPage] tag with the optional -Top and/or -Width parameters. The example below places the inserted page 50 points away from the top and left sides of the new document page frame.

```
[Var:'New_PDF'=(PDF_Doc: -File='MyFile.pdf',  
                    -Format='PDF',  
                    -Size='A4'))]  
[Var:'Old_PDF'=(PDF_Read:-File='/documents/somepdf.pdf')]  
[$New_PDF->(InsertPage: $Old_PDF, 1, -Width=50, -Height=50)]
```

Accessing PDF File Information

Parameter values of a [PDF_Doc] variable can be returned using special accessor tags. These tags return specific values such as the page size, margin size, or the value of any other [PDF_Doc] variable described in the previous section. All PDF accessor tags in LDML 7 are defined in *Table 6: PDF Accessor Tags*.

Table 6: PDF Accessor Tags

Tag	Description
[PDF_Doc->GetMargins]	Returns the current page margins as an array data type (Array: Left, Right, Top, Bottom).
[PDF_Doc->GetSize]	Returns the current page size as an array of width and height point values (Array: Width, Height).
[PDF_Doc->GetColor]	Returns the current color as a hex string.
[PDF_Doc->GetHeaders]	Returns all document headers as a map data type (Map: 'Header1'='Content1', 'Header2'='Content2', ...).
[PDF_Doc->SetFont]	Sets a font for all following text. The value is a [PDF_Font] object.

To return PDF page margins:

Use the [PDF_Doc->GetMargins] tag. The following example returns the current margins of a defined [PDF_Doc] variable.

```
[$MyFile->GetMargins] → (Array: 72.0, 72.0, 72.0, 72.0)
```

To return a PDF page size:

Use the [PDF_Doc->GetSize] tag. The following example returns the current sizes of a defined [PDF_Doc] variable.

```
[$MyFile->GetSize] → (Array: 468.0, 648.0)
```

To return a PDF base font color:

Use the [PDF_Doc->GetColor] tag. The following example returns the base font color of a defined [PDF_Doc] variable.

```
[$MyFile->GetColor] → #333333
```

Saving PDF Files

Once a [PDF_Doc] variable has been filled with the desired content, the [PDF_Doc->Close] tag must be used to signal that the PDF file is finished and is ready to be written to file or served.

Table 7: [PDF_Doc->Close] Tag

Tag	Description
[PDF_Doc->Close]	Closes [PDF_Doc] variable and commits it to file after all desired data has been added to it. Additional data may not be added to the specified variable after this tag is used.

To close a PDF file:

Use the [PDF_Doc->Close] tag after all desired modifications have been performed on the [PDF_Doc] variable.

```
[Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',  
                    -Format='PDF',  
                    -Size='A4',  
                    -Margin=(Array: 144.0, 144.0, 72.0, 72.0))]  
[Var:'Font'=(PDF_Font: -Face='Helvetica', -Size=36)]  
[Var:'Text'=(PDF_Text:'I am a PDF document', -Font=$Font)]  
[$MyFile->(Add: $Text)]  
[$MyFile->Close]
```

Creating Text Content

Text content is the most basic type of data within a PDF document. PDF text is first defined as a [PDF_Text] object, and then added to a PDF variable using the [PDF_Doc->Add] tag.

[PDF_Text] objects may be positioned within the current PDF page using the -Left and -Top parameters of the [PDF_Doc->Add] tag. Otherwise, if no positioning parameters are specified, the text will be added to the top left corner of the page by default.

Using Fonts

Before adding text, it is important to first define the font and style for the text to determine how it will appear. This is done using the [PDF_Font] tag.

Table 8: PDF Font Tag and Parameters

Tag	Description
[PDF_Font]	Stores all the specifications for a font style. This include font family, size, style, and color. Parameters are used with the [PDF_Font] tag that define the font family, size, color, and specifications. The following parameters may be used with the [PDF_Font] tag.
-Face	Specifies the font by its family name. Allowed font names are Courier, Courier-Bold, Courier-Oblique, Courier-BoldOblique, Helvetica, Helvetica-Bold, Helvetica-Oblique, Helvetica-BoldOblique, Symbol, Times-Roman, Times-Bold, Times-Italic, Times-BoldItalic, and ZapfDingbats.
-File	Creates a font from a local font file. The file name and path to the font must be specified (e.g /Fonts/Courier.ttf). This parameter may be used instead of the -Face parameter. Optional.
-Size	Sets the font size in points. Requires an integer point value as a parameter (e.g 14). Optional.
-Color	Sets the font color. Require a hex color string as a parameter (e.g '#550000'). Defaults to '#000000' if not used. Optional.
-Encoding	Sets the desired font encoding. The font encoding defaults to 'CP1252' if not specified. TrueType fonts can be asked to return an array of supported encodings via the [PDF_Font->GetSupportedEncodings] member tag. Optional.
-Embed	Embeds the fonts used within the PDF document as opposed to relying on the client PDF reader for font information. Optional.

The following examples show how to set variables as [PDF_Font] types that define the font styles that are used in a PDF document.

To set a basic font style:

Set a variable as a [PDF_Font] tag. The following example sets a style to be a standard Helvetica font with a size of 14 points. The font color is green.

```
[Var:'Font1'=(PDF_Font: -Face='Helvetica',
                        -Size=14,
                        -Color='#005500')]
```

Individual parameters may be viewed and changed in a [PDF_Font] variable using [PDF_Font] member tags. These parameters are most useful for retrieving information about a [PDF_Font] object that was defined using the -File parameter, and are summarized in *Table 9: [PDF_Font] Member Tags*.

Table 9: [PDF_Font] Member Tags

Tag	Description
[PDF_Font->SetFace]	Changes the font face of the [PDF_Font] variable to one of the allowed font names.
[PDF_Font->SetColor]	Changes the font color of the [PDF_Font] variable.
[PDF_Font->SetSize]	Changes the font size of the [PDF_Font] variable.
[PDF_Font->SetEncoding]	Changes the encoding of the [PDF_Font] variable.
[PDF_Font->SetUnderline]	Sets the [PDF_Font] variable style to underlined. Requires a boolean parameter of 'True' if used.
[PDF_Font->GetFace]	Returns the current font face of a [PDF_Font] variable.
[PDF_Font->GetColor]	Returns the current font color of a [PDF_Font] variable.
[PDF_Font->GetSize]	Returns the current font size of a [PDF_Font] variable.
[PDF_Font->GetEncoding]	Returns the current encoding of a [PDF_Font] variable.
[PDF_Font->GetPSFontName]	Returns the exact Postscript font name of the current font of a [PDF_Font] variable (e.g. AdobeCorlDMinBd).
[PDF_Font->IsTrueType]	Returns True if the current font is a True Type font.
[PDF_Font->GetSupportedEncodings]	Returns an array of all supported encodings for a current True Type font face (Array:'1252 Latin 1','1253 Greek').

[PDF_Font->GetFullFontName]	Returns the full True Type name of the current font of a [PDF_Font] variable (e.g Comic Sans MS Negreta).
[PDF_Font->TextWidth]	Returns an integer value representing how wide (in pixels) the text would be using the current [PDF_Font] variable. Requires a string value, which is the text to return the width of.

To change a font face:

Use the [PDF_Font->SetFace] tag. The following example sets a defined [PDF_Font] variable to a standard Courier font.

```
[$MyFont->(SetFace:'Courier')]
```

To change a font color:

Use the [PDF_Font->SetColor] tag. The following example sets a defined [PDF_Font] variable to the color red.

```
[$MyFont->(SetColor:'#990000')]
```

To underline a font:

Use the [PDF_Font->SetUnderline] tag. The following example sets a predefined [PDF_Font] variable to use an underlined style.

```
[$MyFont->(SetUnderline: 'True')]
```

To return a font face:

Use the [PDF_Font->GetFace] tag. The following example returns the current font face of a defined [PDF_Font] variable.

```
[$MyFont->GetFace] → Courier
```

To return a font encoding:

Use the [PDF_Font->GetEncoding] tag. The following example returns the encoding of the current font face of a defined [PDF_Font] variable.

```
[$MyFont->GetEncoding] → CP1252
```

Adding Text

PDF text content is constructed using the [PDF_Text] tag, which is then added to a [PDF_Doc] variable using the [PDF_Doc->Add] tag. The [PDF_Text] constructor tag and parameters are described below.

Table 10: [PDF_Text] Tag and Parameters

Tag	Description
[PDF_Text]	Creates a text object to be added to a [PDF_Doc] variable. Requires the text string to be added to the PDF document as the first parameter. Optional parameters are listed below.
-Type	Specifies the text type. This can be 'Chunk', 'Phrase', or 'Paragraph'. Different parameters are available for each of these types, as described below. Defaults to the 'Paragraph' type if no -Type parameter is specified. Optional.
-Color	Sets the font color. Requires a hex color string as a parameter (e.g. '#550000'). Defaults to '#000000' if not used. Optional.
-BackgroundColor	Sets the text background color. Require a hex color string as a parameter (e.g. '#550000'). Optional.
-Underline	Keyword parameter underlines the text. Optional.
-TextRise	Sets the baseline shift for superscript. Requires a decimal value that specifies the text rise in points. Optional.
-Font	Sets the font for the specified text. The value is a [PDF_Font] variable, which is described in the <i>Using Fonts</i> section of this chapter. The font defaults to the current inherited font if no -Font parameter is specified.
-Anchor	Links the specified text to a URL. The value of the parameter is the URL string (e.g. 'http://www.example.com'). Optional.
-Name	Sets the name of an anchor destination within a page. The value of the parameter is the anchor name (e.g. 'Name'). Optional.
-GoTo	Links the specified text to a local anchor destination to go to. The value of the parameter is the local anchor name (e.g. 'Name'). Optional.
-File	Links the specified text to a PDF document. The value of the parameter is a PDF file name (e.g. 'Somefile.pdf'). The -Goto parameter can be used concurrently to specify an anchor name within the destination document. Optional.
-Leading	Sets the paragraph leading space in points (the space above and below the text), and requires a decimal value. For 'Phrase' and 'Paragraph' types only.
-Align	Sets the alignment of the text in the page ('Left', 'Center', or 'Right'). Optional.

-IndentLeft	Sets the left indent of the text object. Requires a decimal value which is the number of points to indent the text. Optional. Available for 'Paragraph' types only.
-IndentRight	Sets the right indent of the text object. Requires a decimal value which is the number of points to indent the text. Optional. Available for 'Paragraph' types only.

The following examples show how to add text to a defined PDF variable named MyFile that has been initialized previously using the [PDF_Doc] tag.

To add a chunk of text:

Use the [PDF_Text] tag with the -Type='Chunk' parameter. The following example adds the text Blue World to the [PDF_Doc] variable with a predefined font. The text is positioned in the top left corner of the page by default.

```
[Var:'Text'=(PDF_Text:'Blue World', -Type='Chunk', -Font=$MyFont)]
[$MyFile->(Add: $Text)]
```

To add a paragraph of text:

Use the [PDF_Text] tag with the -Type='Paragraph' parameter. The following example adds three sentences of text to the [PDF_Doc] variable with a predefined font.

```
[Var:'Text'=(PDF_Text:'The mysterious file cabinet in orbit has been successfully
lassoed. The file cabinet had been traveling at a velocity of 300 meters per second.
Top scientists suspect that the cabinet had been in orbit for some time.',
-Type='Paragraph', -Font=$MyFont, -Leading=10.0, -IndentLeft=20.0)]
```

To add a linked phrase:

Use the [PDF_Text] tag with the -Anchor parameter. The following example adds the text Click here to go somewhere to the [PDF_Doc] variable with a predefined font, and links the phrase to <http://www.example.com>.

```
[Var:'Text'=(PDF_Text:'Click here to go somewhere', -Type='Chunk', -Font=$MyFont,
-Ancor='http://www.example.com', -Underline=true)]
[$MyFile->(Add: $Text, -Left=100.0, -Top=100.0)]
```

Adding Floating Text

Instead of adding text to the flow of the page, text can also be positioned on a page using the [PDF_Doc->DrawText] tag. The [PDF_Doc->DrawText] tag accepts coordinates that allow the text to be placed at an absolute position on the page.

Table 11: [PDF_Doc->DrawText] Tag

Tag	Description
[PDF_Doc->DrawText]	Adds specified text that is positioned on a page using point coordinates. A required -Leading parameter sets the text leading space in points (the space above and below the text), and requires a decimal value. A -Left parameter specifies the placement of the left side of the text from the left side of the page in points, and a -Top parameter specifies the placement of the bottom of the image from the bottom of the page in points (decimal value).

To add floating text:

Use the [PDF_Doc->DrawText] tag. The following example adds the text Some floating text to the [PDF_Doc] variable with a predefined font at the coordinates specified in the -Top and -Left parameters. The coordinates represent the distance in points from the lower and left sides of the page.

```
[$MyFile->(DrawText:'Some floating text', -Font=$MyFont,
                                     -Left=144.0, -Top=480.0)
```

Adding Lists

A list of items can be constructed using the [PDF_List] tag, which can be added to a [PDF_Doc] variable. The [PDF_List] constructor tag and parameters are described below.

Table 12: [PDF_List] Tags and Parameters

Tag	Description
[PDF_List]	Creates a list object to be added to a [PDF_Doc] variable. Text list items are added to this object using the [PDF_List->Add] tag. Optional parameters for this object are described below.
-Format	Specifies whether the list is numbered, lettered, or bulleted. Requires a value of 'Number', 'Letter', 'Bullet'. Defaults to 'Bullet' if no -Format parameter is specified. Optional.
-Bullet	Specifies a custom character to use as the bullet character. Requires a character as a parameter (e.g. 'x'). Defaults to '.' if not specified. Optional.
-Indent	Sets the space between the bullet and the list item. Requires a decimal or integer parameter which is the width of the indentation in points. Optional.

-Font	Sets the font for the specified text. The value is a [PDF_Font] variable, which is described in the <i>Using Fonts</i> section of this chapter. The font defaults to the current inherited font if no -Font parameter is specified.
-Align	Sets the alignment of the list in the page ('Left', 'Center', or 'Right'). Optional.
-Color	Sets the font color. Requires a hex color string as a parameter (e.g. '#550000'). Defaults to '#000000' if not used. Optional.
-BackgroundColor	Sets the text background color. Require a hex color string as a parameter (e.g. '#550000'). Optional.
-Leading	Sets the list leading space in points (the space above and below the text), and requires a decimal value. Optional.
[PDF_List->Add]	Add objects to the list. Requires a text string or a [PDF_Text] object as a parameter.

To add a numbered list:

Use the [PDF_List] tag with the -Format='Number' parameter to define the list, and the [PDF_List->Add] tag to add items to the list. The example below creates a numbered list with three items.

```
[Var:'List'=(PDF_List: -Format='Number', -Align='Center', -Font=$MyFont)]
[$List->(Add:'This is item one')]
[$List->(Add:'This is item two')]
[$List->(Add:'This is item three')]
[$MyFile->(Add: $List, -Top=400.0)]
```

To add a bulleted list:

Use the [PDF_List] tag with the -Format='Number' parameter to define the list, and the [PDF_List->Add] tag to add items to the list. The example below adds a numbered list with four items, where a hyphen (-) is used as the bullet character.

```
[Var:'List'=(PDF_List: -Format='Bullet', -Bullet='-', -Font=$MyFont)]
[$List->(Add:'This is item one')]
[$List->(Add:'This is item two')]
[$List->(Add:'This is item three')]
[$List->(Add:'This is item four')]
[$MyFile->(Add: $List, -Top=400.0)]
```

Special Characters

When adding text to a [PDF_Doc] object, special characters can be used to designate lines breaks, tabs, and more. These characters are summarized in *Table 13: Special Characters*.

Table 13: Special Characters

Character	Description
\n	Line break character (Mac OS X and Linux).
\r\n	Line break character (Windows).
\t	Tab character.
\"	Double quote character.
\'	Single quote character.
\\	Backslash character.

To use special characters in a text string:

The following example shows how to use special characters within a [PDF_Doc] text tag.

```
[$MyFile->(Add: "\\ \t \'Single Quotes\' \'"Double Quotes'" \t \\", -Font=$MyFont)]
```

Creating and Using Forms

Forms can be created in PDF documents for submitting information to a Web site. PDF forms use the same attributes as HTML forms, making them useful for submitting information to a Web site in place of an HTML form. This section describes how to create form elements within a PDF file, and also how PDF forms can be used to submit data to a Lasso-enabled database.

Note: Due to the iText implementation of PDF support in Lasso Professional 7, PDF documents created may contain one form only.

Creating Forms

Form elements are created in [PDF_Doc] variables using [PDF_Doc] form member tags, which are listed in *Table 14: [PDF_Doc] Form Member Tags*.

Table 14: [PDF_Doc] Form Member Tags

Tag	Description
[PDF_Doc->AddTextField]	Adds a text field to a form. A required -Name parameter specifies the name of the text field, and a required -Value parameter specifies the default value entered. A required -Font parameter is used to specify a [PDF_Font] variable for the text font.
[PDF_Doc->AddPasswordField]	Adds a password field to a form. A required -Name parameter specifies the name of the password field, and a required -Value parameter specifies the default value entered. A required -Font parameter is used to specify a [PDF_Font] variable for the text font.
[PDF_Doc->AddTextArea]	Adds a text area to a form. A required -Name parameter specifies the name of the text area, and a required -Value parameter specifies the default value entered. A required -Font parameter is used to specify a [PDF_Font] variable for the text font.
[PDF_Doc->AddCheckBox]	Adds a check box to a form. A required -Name parameter specifies the name of the checkbox, and a required -Value parameter specifies the value for the checkbox. An optional -Checked parameter specifies that the checkbox is checked by default.
[PDF_Doc->AddRadioGroup]	Adds a radio button group to a form. A required -Name parameter specifies the name of the radio button group. Radio buttons must be assigned to the group using the [PDF_Doc->AddRadioButton] tag.
[PDF_Doc->AddRadioButton]	Adds a radio button to a form. A required -Group parameter specifies the name of the radio button group, and a required -Value parameter specifies the value of the radio button.
[PDF_Doc->AddComboBox]	Adds a pull-down menu to a form. A required -Name parameter specifies the name of the pull-down menu, and a required -Values parameter specifies the array of values contained in the menu (Array: 'Value1', 'Value2'). An -Options parameter may be used instead of the -Values parameter that specifies a pair for each value. The first element in the pair is the value to be used upon form submission, and the second element is the human-readable label to be used for display only. An optional -Default parameter specifies the name of a default value selected. An optional -Editable parameter specifies that the user may edit the values on the menu. A required -Font parameter is used to specify a [PDF_Font] variable for the text font.

[PDF_Doc->AddSelectList]	Adds a select list to a form. A required -Name parameter specifies the name of the select list, and a required -Values parameter specifies the array of values contained in the select list (Array: 'Value1', 'Value2'). An -Options parameter may be used instead of the -Values parameter that specifies a pair data type for each value. The first element in the pair is the value to be used upon form submission, and the second element is the human-readable label to be used for display only. An optional -Default parameter specifies the name of a default value selected. A required -Font parameter is used to specify a [PDF_Font] variable for the text font.
[PDF_Doc->AddHiddenField]	Adds a hidden field to a form. A required -Name parameter specifies the name of the hidden field, and a -Value parameter specifies the default value entered.
[PDF_Doc->AddSubmitButton]	Adds a submit button to a form. Also specifies the URL to which the form data will be submitted. A required -Name parameter specifies the name of the button, and a required -Value parameter specifies the name displayed on the button. A required -URL parameter specifies the URL of the response page. A -Font parameter is used to specify a [PDF_Font] variable for the button text font, and an optional -Caption parameter specifies a caption (displayed name) for the button.
[PDF_Doc->AddResetButton]	Adds a reset button to a form. A required -Name parameter specifies the name of the button, and a required -Value parameter specifies the name displayed on the button. A -Font parameter is used to specify a [PDF_Font] variable for the button text font, and an optional -Caption parameter specifies a caption (displayed name) for the button.

Field Label Note: With the exception of the [PDF_Doc->AddSubmitButton] and [PDF_Doc->AddSubmitButton] tags, no form input element tags include captions or labels with the field elements. Field captions and labels can be applied using the [PDF_Text] and [PDF_Doc->Add] tags to position text appropriately. See the *Creating Text Content* section for more information.

All [PDF_Doc] form member tags, with the exception of [PDF_Doc->AddHiddenField], require placement parameters for specifying the exact positioning of form elements within a page. These parameters are summarized in *Table 15: Form Placement Parameters*.

Table 16: Form Placement Parameters

Tag	Description
-Left	Specifies the placement of the left side of the form element from the left side of the current page in points. Requires a decimal value. Optional.
-Top	Specifies the placement of the bottom of the form element from the bottom of the current page in points. Requires a decimal value. Optional.
-Width	Specifies the width of the form element in points. Requires a decimal value. Optional.
-Height	Specifies the height of the form element in points. Requires a decimal value. Optional.

To add a text field:

Use the [PDF_Doc->AddTextField] tag. The example below adds a field named Field_Name that has Some Text entered by default. The field size is 144.0 points (two inches) wide and 36.0 points high.

```
[MyFile->(AddTextField: -Name='Field_Name',
                    -Value='Some Text',
                    -Font=$MyFont,
                    -Left=72.0, -Top=350.0, -Width=144.0, -Height=36.0))]
```

To add a text area:

Use the [PDF_Doc->AddTextArea] tag. The example below adds a text area named Field_Name that has the text Insert default text here entered by default. The field size is 144.0 points wide and 288.0 points high.

```
[MyFile->(AddTextArea: -Name='Field_Name',
                    -Value='Insert default text here',
                    -Font=$MyFont,
                    -Left=72.0, -Top=300.0, -Width=144.0, -Height=288.0))]
```

To add a checkbox:

Use the [PDF_Doc->AddCheckbox] tag. The example below adds a field named Field_Name with a checked value of Checked_Value that is checked by default. The checkbox is 4.0 points wide and 4.0 points high, and is positioned 272.0 points from the bottom and left sides of the page.

```
[MyFile->(AddCheckBox: -Name='Field_Name',
                    -Value='Checked_Value',
                    -Checked,
                    -Left=272.0, -Top=272.0, -Width=4.0, -Height=4.0))]
```

To add a group of radio buttons:

Use the [PDF_Doc->AddRadioGroup] and [PDF_Doc->AddRadioButton] tags. The example below adds a radio button group named Group_Name, and adds two radio buttons with the values of Yes and No. The radio buttons are 6.0 points wide and 6.0 points high each.

Note: If the [PDF_Doc->AddRadioGroup] tag is not used, then radio buttons will not appear in the form.

```
[MyFile->(AddRadioGroup: -Name='Group_Name')]
[MyFile->(AddRadioButton: -Group='Group_Name',
                        -Value='Yes',
                        -Left=72.0, -Top=372.0, -Width=6.0, -Height=6.0)]
[MyFile->(AddRadioButton: -Group='Group_Name',
                        -Value='No',
                        -Left=90.0, -Top=372.0, -Width=6.0, -Height=6.0)]
```

To add an editable pull-down menu:

Use the [PDF_Doc->AddComboBox] tag. The example below adds a pull-down menu named Menu_Name with the values One, Two, Three, and Four as menu values. The value One is selected by default, and an -Editable parameter allows the users to edit the values if desired. The pull-down menu size is 144.0 points wide and 36.0 points high.

```
[MyFile->(AddComboBox: -Name='List_Name',
                      -Values=(Array: 'One', 'Two', 'Three', 'Four'),
                      -Default='One',
                      -Editable,
                      -Left=72.0, -Top=272.0, -Width=144.0, -Height=36.0)]
```

To add a pull-down menu with different displayed values:

Use the [PDF_Doc->AddComboBox] tag with the -Options parameter instead of the -Values parameter. The example below adds a pull-down menu named Menu_Name with the values 1, 2, 3, and 4 as submitable menu values, but displays the names One, Two, Three, and Four for each value. No value is selected by default.

```
[MyFile->(AddComboBox: -Name='List_Name',
                      -Values=(Array: (Pair: (1)=(One)),
                                      (Pair: (2)=(Two)),
                                      (Pair: (3)=(Three)),
                                      (Pair: (4)=(Four))),
                      -Left=72.0, -Top=272.0, -Width=144.0, -Height=36.0)]
```

To add a select list:

Use the [PDF_Doc->AddSelectList] tag. The example below adds a select list named List_Name with the values One, Two, Three, and Four as list items. The select list is 144.0 points wide and 288.0 points high, and is positioned 72.0 points from the bottom and left sides of the page.

```
[MyFile->(AddSelectList: -Name='List_Name',
                        -Values=(Array: 'One', 'Two', 'Three', 'Four'),
                        -Default='One',
                        -Left=72.0, -Top=72.0, -Width=144.0, -Height=288.0)]
```

To add a hidden field:

Use the [PDF_Doc->AddHiddenField] tag. The example below adds a hidden field named Field_Name with a value of Hidden_Value to a [PDF_Doc] variable named MyFile. No placement coordinates are needed because the field is not displayed on the page.

```
[MyFile->(AddHiddenField: -Name='Field_Name',
                          -Value='Some_Value')]
```

To add a submit button:

Use the [PDF_Doc->AddSubmitButton] tag. The example below adds a submit button named Button_Name with a value of Submitted_Value. The -URL parameter specifies that the user will be taken to <http://www.example.com/responsepage.lasso> when the button is selected in the form. A -Caption parameter specifies the displayed name of the button, which is Submit This Form.

```
[MyFile->(AddSubmitButton: -Name='Button_Name',
                          -Value='Submitted_Value',
                          -URL='http://www.example.com/responsepage.lasso',
                          -Caption='Submit This Form',
                          -Left=72.0, -Top=72.0, -Width=144.0, -Height=36.0)]
```

To add a reset button:

Use the [PDF_Doc->AddResetButton] tag. The example below adds a submit button named Button_Name with a value of Submitted_Value. A -Caption parameter specifies the displayed name of the button, which is Reset This Form.

```
[MyFile->(AddResetButton: -Name='Button_Name',
                          -Value='Submitted_Value',
                          -Caption='Reset This Form',
                          -Left=72.0, -Top=72.0, -Width=144.0, -Height=36.0)]
```


Submitting Form Data to Lasso-Enabled Databases

In Lasso Professional 7, one has the ability to submit data from a PDF form to a Lasso-enabled database. PDF forms may be used in the same way as HTML forms to submit action parameters to an LDML response page, where database actions can occur via an [Inline] tag.

The following example shows the HTML form example in *Chapter 6: Database Interaction Fundamentals > Inline Method* as it would appear in a [PDF_Doc] variable in a Form.lasso page.

To submit information to database using a PDF form:

- 1 In the Form.lasso page, name the PDF form fields to correspond to the names of fields in the desired database. The names of these fields will be used in the [Inline] tag in the LDML response page.

```
[Var:'MyFile'=(PDF_Doc: -File='Form.pdf', -Size='A4')]
[Var:'MyFont'=(PDF_Font: -Face='Helvetica', -Size=12)]
[$MyFile->(DrawText: 'First Name:', -Font=$MyFont, -Left=80.0, -Top=60.0)]
[$MyFile->(DrawText: 'Last Name:', -Font=$MyFont, -Left=80.0, -Top=60.0)]
[$MyFile->(AddTextField: -Name='First Name',
                        -Value='Enter First Name',
                        -Left=144.0, -Top=72.0, -Width=144.0, -Height=36.0)]
[$MyFile->(AddTextField: -Name='Last Name',
                        -Value='Enter Last Name',
                        -Left=144.0, -Top=92.0, -Width=144.0, -Height=36.0)]
```

- 2 Create a submit button in the Form.lasso page that contains the name and URL of the LDML response page.

```
[$MyFile->(AddSubmitButton: -Name='Search',
                           -Value='Search',
                           -Caption='Click here to Search',
                           -URL='http://www.example.com/Response.lasso',
                           -Font=$MyFont)]

[$MyFile->Close]
```

After the [PDF_Doc] variable is closed and executed on the server, a Form.pdf file will be created with a form.

- 3 In the Response.lasso page, create an [Inline] tag that uses the action parameters passed from the PDF form to perform a database action. This example performs a search on the Contacts database using the First_Name and Last_Name parameters passed from the PDF form.

```
[Inline: -Search,
        -Database='Contacts',
        -Table='People',
        -KeyField='ID',
        'First_Name'=(Action_Param: 'First_Name'),
```

```
'Last_Name'=(Action_Param: 'Last_Name'))
There were [Found_Count] record(s) found in the People table.
[Records]
<br>[Field: 'First_Name'] [Field: 'Last_Name']
[/Records]
[/Inline]
```

If the user of the PDF form entered Jane for the first name and Doe for the last name, then the following results would be returned.

→ There were 1 record(s) found in the People table.
Jane Doe

Creating Tables

Tables can be created in PDF documents for displaying data. These are created using the [PDF_Table] tag, and added to a PDF variable using [PDF_Doc] member tags, which are described in this section.

Defining Tables

Tables for organizing data can be defined for use in a PDF document using the [PDF_Table] tag. This tag is used to set a variable as a [PDF_Table] type, and the [PDF_Table] variable is then added to a [PDF_Doc] variable.

Table 17: [PDF_Table] Tag and Parameters

Tag	Description
[PDF_Table]	Creates a table to be placed in a PDF. Uses parameters which set the basic specifications of the table to be created.
-Cols	Specifies the number of columns in a table. Required.
-Rows	Specifies the number of rows in a table. Required.
-Spacing	Specifies the spacing around a table cell. Defaults to 0 (no spacing) if not specified. Optional.
-Padding	Specifies the padding within a table cell. Defaults to 0 (no padding) if not specified. Optional.
-Width	Specifies the width of the table as a percentage of the current page width. Defaults to the width of the cell text plus spacing, padding, and borders if not specified. Optional.
-BorderWidth	Specifies the border width of the table in points. Requires a decimal value. Optional.

-BorderColor	Specifies the border color of the table. Requires a hex color string (e.g. '#000000'). Optional.
-BackgroundColor	Specifies the background color of the table. Requires a hex color string (e.g. '#CCCCCC'). Optional.
-ColWidth	Sets the column width for each column in the table. Requires an array of decimals representing the width percentage of each column. Optional.

Member tags can be used to set additional specifications for a [PDF_Table] variable, as well as access parameter values from [PDF_Table] variables. These tags are summarized in *Table 18: [PDF_Table] Member Tags*.

Table 18: [PDF_Table] Member Tags

Tag	Description
[PDF_Table->GetColumnCount]	Returns the number of columns in a [PDF_Table] variable.
[PDF_Table->GetRowCount]	Returns the number of rows in a [PDF_Table] variable.
[PDF_Table->GetAbsWidth]	Returns the total [PDF_Table] variable width in pixels.

To create a basic table:

Use the [PDF_Table] tag. The example below creates a table with two columns and five rows, with table cell spacing of one point and cell padding of two points. The width of the table is set at 75 percent of the current page width.

```
[Var:'MyTable'=(PDF_Table: -Cols=2,
                           -Rows=5,
                           -Spacing=1,
                           -Padding=2,
                           -Width=75,
                           -BackgroundColor='#CCCCCC')]
```

To create a table with a border:

Use the [PDF_Table] tag with the -Border... parameters. The example below creates a basic table, and then adds a black border with a width of 3 points to the table.

```
[Var:'MyTable'=(PDF_Table: -Cols=2,
                           -Rows=5,
                           -Spacing=1,
                           -Padding=2)]
-BorderWidth=3,
-BorderColor='#000000')]
```

To rotate a table:

Use the [PDF_Table] tag with the -Rotate parameter. The example below creates a basic table, and then rotates it by 90 degrees clockwise.

```
[Var:'MyTable'=(PDF_Table: -Cols=2,
                        -Rows=5,
                        -Spacing=1,
                        -Padding=2,
                        -Rotate=90)]
```

To create a table with pre-specified column widths:

Use the [PDF_Table] tag with the -ColWidth parameter. The example below creates a basic table with percentage widths for three columns.

```
[Var:'MyTable'=(PDF_Table: -Cols=2,
                        -Rows=5,
                        -Spacing=1,
                        -Padding=2,
                        -ColWidth=(Array: '50.0', '25.0', '25.0'))]
```

Adding Content to Table Cells

Content is added to table cells using additional [PDF_Table] member tags. These tags are summarized in *Table 19: Cell Content Tags*.

Table 19: Cell Content Tags

Tag	Description
[PDF_Table->Add]	Inserts text content or a new nested table into a cell. Requires a text string or a new [PDF_Table] variable to be inserted as a parameter. Also requires parameters described in the following table.
-Col	Specifies the column number starting from 0 (numbered from left to right) of the cell to add or remove. Requires an integer value. Required.
-Row	Specifies the row number starting from 0 (numbered from top to bottom) of the cell to add or remove. Requires an integer value. Required.
-Colspan	Specifies the number of columns a cell should span. Requires an integer value. Required.
-Rowspan	Specifies the number of rows a cell should span. Requires an integer value. Required.
-VerticalAlignment	Vertical alignment for text within a cell. Accepts a value of 'Top', 'Center', or 'Bottom'. Defaults to 'Center' if not specified. Optional.

-HorizontalAlignment	Horizontal alignment for text within a cell. Accepts a value of 'Left', 'Center', or 'Right'. Defaults to 'Center' if not specified. Optional.
-BorderColor	Specifies the border color for the cell (e.g. '#440000'). Defaults to '#000000' if not specified. Optional.
-BorderWidth	Specifies the border width of the cell in points. Requires a decimal value. Defaults to 0 if not specified. Optional.
-Header	Specifies that the cell is a table header. This is typically used for cells in the first row. Optional.
-NoWrap	Specifies that the text contained in a cell should not wrap to conform to the cell size specifications. If used, the cell will expand to the right to accomodate longer text strings. Optional.

To add a cell to a table:

Use the [PDF_Table->Add] tag. The example below adds a cell to the first row and column in a table. Note that the first row and column are numbered 0.

```
[MyTable->(Add: 'This is the first cell in my table', -Col=0, -Row=0, -Colspan=0, -Rowspan=0)]
```

To add a multi-column cell to a table:

Use the [PDF_Table->Add] tag with the number of columns to span for the -Column parameter. The example below adds a cell to the first row that spans three columns. The -NoWrap parameter is used to indicate that the added text will not be wrapped into multiple lines.

```
[MyTable->(Insert: 'This text will only stay on one line regardless of the table size', -Col=0, -Row=0, -Colspan=3, -Rowspan=1, -NoWrap)]
```

To add a header cell to a table:

Use the [PDF_Table->Add] tag with the -Header parameter. The example below adds the header My Column Title to the first column of the table.

```
[MyTable->(Add: 'My Column Title', -Col=0, -Row=0, -Colspan=0, -Rowspan=0, -Header)]
```

To add a cell with a border to a table:

Use the [PDF_Table->Add] tag with the -BorderWidth and -BorderColor parameter. The example below adds a cell with a red border to the first column of the table.

```
[MyTable->(Add: 'This cell has a border', -Col=0, -Row=0, -Colspan=0, -Rowspan=0, -BorderWidth=45.0, -BorderColor='#440000']
```

Adding Tables

Once a [PDF_Table] object is completely defined and has cell content, it may then be added to a [PDF_Doc] objects using the [PDF_Doc->Add] tag.

To add a table to a [PDF_Doc] variable:

Use the [PDF_Doc->Add] tag. The following example adds a predefined [PDF_Table] variable named MyTable to a [PDF_Doc] variable named MyFile.

```
[$MyFile->(Add: $MyTable)]
```

Creating Graphics

This section describes how to draw custom graphic objects and insert image files within a PDF document.

Inserting Images

Image files can be placed within PDF pages via the [PDF_Doc->AddImage] tag, which is summarized in *Table 20: [PDF_Doc] Image Tag*.

Table 20: [PDF_Image] Tag and Parameters

Tag	Description
[PDF_Image]	Casts an image file as an LDML object so it can be placed in a PDF file. Requires either a -File, -URL, or -Raw parameter, as described below. Only images in JPEG, GIF, PNG, and WMF formats may be used.
-File	Specifies the local path to an image file. Required if the -URL or -Raw parameters are not used.
-URL	Specifies a URL to an image file. Required if the -File or -Raw parameters are not used.
-Raw	Inputs a raw string of bits representing the image. Required if the -URL or -File parameters are not used.
-Height	Scales the image to the specified height. Requires a decimal value which is the desired image height in points. Optional.
-Width	Scales the image to the specified width. Requires a decimal value which is the desired image width in points. Optional.
-Proportional	Keyword parameter specifying that all scaling should preserve the aspect ratio of the inserted page. Optional.
-Rotate	Rotates the image by the specified degrees clockwise. Optional.

To add an image file to a [PDF_Doc] variable:

Use the [PDF_Image] tag. The following example adds a file named Image.jpg in a /Documents/Images/ folder to a [PDF_Doc] variable named MyFile.

```
[Var:'Image'=PDF_Image: -File='/Documents/Images/Image.jpg']
[$MyFile->(Add: $Image, -Left=144.0, -Top=300.0)]
```

To scale image file:

Use the [PDF_Image] tag with the -Height or -Width parameter. The following example proportionally reduces the size of the added image by 50%.

```
[Var:'Image'=PDF_Image: -File='/Documents/Images/Image.jpg', -Height=50.0]
[$MyFile->(Add: $Image, -Left=144.0, -Top=300.0)]
```

To rotate an image file :

Use the [PDF_Image] tag with the -Rotate parameter. The following example rotates the added image by 90 degrees clockwise.

```
[Var:'Image'=PDF_Image: -File='/Documents/Images/Image.jpg', -Rotate=90.0]
[$MyFile->(Add: $Image, -Left=144.0, -Top=300.0)]
```

Drawing Graphics

To draw custom graphics, Lasso uses a coordinate system to determine the placement of each graphical object. This coordinate system is a standard coordinate plane with horizontal (X) vertical (Y) axis, where a point on a page is defined by an array containing horizontal and vertical position values (X, Y). The base point of the coordinate plane (0, 0) is located in the lower left corner for the current page. Increasing an X-Value moves a point to the right in the page, and increasing the Y-Value moves the point up in the page. The maximum X and Y values are defined by the current width and height of the page in points.

Custom graphics may be drawn in PDF pages using [PDF_Doc] drawing member tags. These member tags operate by controlling a “virtual pen” which draws graphics similar to a true graphics editor. These member tags are summarized in *Table 21: [PDF_Doc] Drawing Member Tags*.

Table 21: [PDF_Doc] Drawing Member Tags

Tag	Description
[PDF_Doc->SetColor]	Sets the color and style for subsequent drawing operations. A required 'Type' parameter specifies whether the drawing action is of type Stroke, Fill, or Both. A required 'Color' parameter specifies a color type of Gray, RGB, or CMYK. If Gray is specified, a decimal specifies a color strength value. If RGB is specified, three decimal values specify red, green and blue values, respectively. If CMYK is specified, four decimal values specify cyan, magenta, yellow, and black values, respectively. Color values are specified as decimals ranging from 0 to 1.0.
[PDF_Doc->SetLineWidth]	Sets the line width for subsequent drawing actions in points. Requires a decimal point value.
[PDF_Doc->MoveTo]	Moves the virtual pen to the specified coordinates. This tag is required to move the virtual pen into the correct position before drawing a line or a curve. Requires a string of integer X-axis and Y-axis coordinates to designate the starting point (X, Y).
[PDF_Doc->Line]	Draws a line. Requires a string of integer points which specifies the starting point and ending point of the line (X1, Y1, X2, Y2).
[PDF_Doc->CurveTo]	Draws a curve. Requires a string of integer points which specifies the starting point, middle point, and ending point of the curve (X1, Y1, X2, Y2, X3, Y3).
[PDF_Doc->Rect]	Draws a rectangle. Requires a string of X and Y integer points which specifies the lower right corner of the rectangle, and the height and width of the rectangle sides from that coordinate (X, Y, Width, Height).
[PDF_Doc->Circle]	Draws a circle. Requires a string of integer points for the center coordinates and a radius length value (X, Y, R).
[PDF_Doc->Arc]	Draws an arc. Requires a string of integer points for the center coordinates and radius of the invisible circle to which the arc belongs, a starting degree which specifies the degrees of the circle at which the arc starts, and an ending degree which specifies the circle degrees at which the arc ends (X, Y, R, Start, End). Angles start with 0 to the right of the center and increase counter-clockwise.

To set the color and style for a drawing action:

Use the [PDF_Doc->SetColor] tag. The example below sets a color of red for all subsequent drawing action until another [PDF_Doc->SetColor] tag is set.


```
[$MyFile->(SetColor: 'Stroke', 'RGB', 0.9, 0.1, 0.1)]
```

To set the line width of a drawing action:

Use the [PDF_Doc->SetLineWidth] tag. The example below sets a line width of 5 points for all subsequent drawing action until another [PDF_Doc->SetLineWidth] tag is set.

```
[$MyFile->(SetLineWidth: 5.0)]
```

To draw a line:

Use the [PDF_Doc->Line] tag. The example below draws a horizontal line from points (8, 8) to points (32, 32). The [PDF_Doc->MoveTo] tag must first be used to move the virtual pen to the starting point coordinates.

```
[$MyFile->(MoveTo: 8, 8)]
[$MyFile->(Line: 8, 8, 32, 32)]
```

To draw a curve:

Use the [PDF_Doc->CurveTo] tag. The example below draws a curve starting from points (8, 8), peaking at points (32, 32), and ending at points (56, 8). The [PDF_Doc->MoveTo] tag must first be used to move the virtual pen the starting point coordinates.

```
[$MyFile->(MoveTo: 8, 8)]
[$MyFile->(CurveTo: 8, 8, 32, 32, 56, 8)]
```

To draw a rectangle:

Use the [PDF_Doc->Rect] tag. The example below draws a rectangle whose lower left corner is at coordinates (10, 60), has left and right sides that are 50 points long, and has top and bottom sides that are 20 point long.

```
[$MyFile->(Rect: 10, 60, 20, 50)]
```

To draw a circle:

Use the [PDF_Doc->Circle] tag. The example below draws a circle whose center is at coordinates (50, 50) and has a radius of 20 points.

```
[$MyFile->(Circle: 50, 50, 20)]
```

To draw an arc:

Use the [PDF_Doc->Arc] tag. The example below draws an arc whose center is at coordinates (50, 50), has a radius of 20 points, and runs from 0 degrees to 90 degrees from the center.

```
[$MyFile->(Arc: 50, 50, 20, 0, 90)]
```

Creating Barcodes

Barcodes are special device-readable images that can be created in PDF documents using the [PDF_Barcode] tag, and added to a PDF variable using [PDF_Doc] member tags, which are described in this section. Lasso Professional 7 can be used to create the following industry-standard barcodes:

- Code 39 (alphanumeric, ASCII subset)
- Code 39 Extended (alphanumeric, escaped text)
- Code 128
- Code 128 UCC/EAN
- Code 128 Raw
- EAN (8 digits)
- EAN (13 digits)
- POSTNET
- PLANET

Creating Bar Codes

Barcodes can be defined for use in a PDF file using the [PDF_Barcode] tag. This tag is used to set a variable as a [PDF_Barcode] type, and the [PDF_Barcode] variable is added to a [PDF_Doc] variable using member tags.

Table 22: [PDF_Barcode] Tag and Parameters

Tag	Description
[PDF_Barcode]	Creates a barcode image to be placed in a PDF. Uses parameters which set the basic specifications of the barcode to be created.
-Type	Specifies the type of barcode to be created. Available parameters are CODE39, CODE39_EX, CODE128, CODE128_UCC, CODE128_RAW, EAN8, EAN13, POSTNET, and PLANET. Defaults to CODE39 if not specified.
-Code	Specifies the numeric or alphanumeric barcode data. Some formats require specific data strings: EAN8 requires an 8-digit integer, EAN13 requires a 13-digit integer, POSTNET requires a zip code, and Code39 requires uppercase characters. Required.
-Color	Specifies the color of the bars in the barcode. Requires a hex string color value. Defaults to '#000000' if not specified. Optional.

-Supplemental	Adds an additional two or five digit supplemental barcode to EAN8 or EAN13 barcode types. Requires a two or five digit integer as a parameter. Optional.
-GenerateChecksum	Generates a checksum for the barcode. Optional.
-ShowCode39StartStop	Displays start and stop characters (*) in the text for Code 39 barcodes. Optional.
-ShowEANGuardBars	Show the guard bars for EAN barcodes. Optional.
-BarHeight	Sets the height of the bars in points. Requires a decimal value.
-BarWidth	Sets the width of the bars in points. Requires a decimal value.
-BaseLine	Sets the text baseline in points. Requires a decimal value.)
-ShowChecksum	Keyword parameter sets the generated checksum to be shown in the text
-Font	Sets the text font. Requires a [PDF_Font] variable.
-BarMultiplier	Sets the bar multiplier for wide bars. Requires a decimal value.
-TextSize	Sets the size of the text. Requires a decimal value.

To create a barcode:

Use the [PDF_Barcode] tag. The example below creates a basic Code 39 barcode with the data 1234567890, and uses the optional Code 39 start and stop characters (*). The barcode is then added to a [PDF_Doc] variable using [PDF_Doc->Add].

```
[Var:'Barcode'=(PDF_Barcode:
-Type='CODE39',
-Code='1234567890',
-ShowCode39StartStop)]
[$MyPDF->(Add: $Barcode, -Left=150.0, -Top=100.0)]
```

To create a barcode with a specified bar width:

Use the [PDF_Barcode] tag with the -BarWidth parameter. The following example sets a [PDF_Barcode] variable with a bar width of 0.2 points.

```
[Var:'Barcode'=(PDF_Barcode:
-Type='CODE39',
-Code='1234567890',
-BarWidth=0.2)]
[$MyPDF->(Add: $Barcode, -Left=150.0, -Top=100.0)]
```

To create a barcode with a specified bar multiplier:

Use the [PDF_Barcode] tag with the -BarMultiplier parameter. The following example sets a [PDF_Barcode] variable with a bar multiplier constant of 4.0. The barcode is then added to a [PDF_Doc] variable using [PDF_Doc->Add].

```
[Var:'Barcode'=(PDF_Barcode:
-Type='CODE39',
-Code='1234567890',
-BarMultiplier=4.0)]
[$MyPDF->(Add: $Barcode, -Left=150.0, -Top=100.0)]
```

To create a barcode with a specified text size:

Use the [PDF_Barcode] tag with the -TextSize parameter. The following example sets a [PDF_Barcode] variable with a text size of 6 points. The barcode is then added to a [PDF_Doc] variable using [PDF_Doc->Add].

```
[Var:'Barcode'=(PDF_Barcode:
-Type='CODE39',
-Code='1234567890',
-TextSize=6)]
[$MyPDF->(Add: $Barcode, -Left=150.0, -Top=100.0)]
```

To create a barcode with a specified font:

Use the [PDF_Barcode] tag with the -Font parameter. The following example sets a [PDF_Barcode] variable font specified in a [PDF_Font] variable named MyFont. The barcode is then added to a [PDF_Doc] variable using [PDF_Doc->Add].

```
[Var:'Barcode'=(PDF_Barcode:
-Type='CODE39',
-Code='1234567890',
-Font=$MyFont)]
[$MyPDF->(Add: $Barcode, -Left=150.0, -Top=100.0)]
```

Example PDF Files

This section provides complete examples of creating PDF files using the tags described in this chapter. Examples include a two-page PDF file with multiple text styles, a PDF file with a form, a PDF file with a table, a PDF file with drawn graphics, and a PDF file with a barcode.

Special Characters Note: All examples in this section use the Mac OS X line break character `\n` in the text sections. If creating PDF files on the Windows or Linux version of Lasso Professional 7, change all instances of `\n` to `\r\n` for Windows, or `\r` for Linux.

PDF Text Example

The following example creates a PDF file that contains two pages of text with multiple text styles.

```
[Var:'Text_Example'=(PDF_Doc: -File='Text_Example.pdf', -Size='A4')]
[$Text_Example->AddPage]
[$Text_Example->(SetPageNumber: 1)]
[Var:'Font1'=(PDF_Font: -Face='Helvetica', -Size='24', -Color='#990000')]
[Var:'Font2'=(PDF_Font: -Face='Helvetica', -Size='14', -Color='#000000')]
[Var:'Font3'=(PDF_Font: -Face='Helvetica', -Size='14', -Color='#0000CC')]
[Var:'Title'=(PDF_Text: 'Lasso Professional 7', -Type='Chunk', -Font=$Font1)]
[$Text_Example->(Add: $Title, -Number=1)]
[Var:'Text1'=(PDF_Text:'\n\nThe Lasso product line consists of authoring and serving
tools that allow Web designers and Web developers to quickly build and serve
powerful data-driven Web sites with maximum productivity and ease. The product
line includes Lasso Professional for building, serving, and administering data-driven
Web sites, and Lasso Studio for building and testing data-driven Web sites within a
graphical editor.\n\nLasso Professional 7 works with the following data sources:',
-Type='Paragraph', -Leading=15, -Font=$Font2)]
[$Text_Example->(Add: $Text1)]
[Var:'List'=(PDF_List: -Format='Bullet', -Bullet='-', -Font=$Font2, -Indent=30)]
[$List->(Add:'FileMaker Pro')]
[$List->(Add:'MySQL')]
[$List->(Add:'Microsoft SQL Server')]
[$List->(Add:'Frontbase')]
[$List->(Add:'Sybase')]
[$List->(Add:'PostgreSQL')]
[$List->(Add:'DB2')]
[$List->(Add:'Plus many other JDBC-compliant databases')]
[$Text_Example->(Add: $List)]
[Var:'Text2'=(PDF_Text:'\nLasso's innovative architecture provides an industry first
multi-platform, database-independent and open standards approach to delivering
database-driven Web sites firmly positioning Lasso technology within the rapidly
evolving server-side Web tools market. Lasso technology is used at hundreds of
thousands of Web sites worldwide.\n\n', -Type='Paragraph', -Font=$Font2)]
[$Text_Example->(Add: $Text2)]
[Var:'Text3'=(PDF_Text:'Click here to go to the Blue World Web site', -Type='Phrase',
-Font=$Font3, -Underline='true', -Anchor='http://www.blueworld.com')]
[$Text_Example->(Add: $Text3)]
[$Text_Example->(DrawText: (String: $Text_Example->GetPageNumber),
-Font=$Font2, -Top=30, -Left=560)]
[$Text_Example->AddPage]
[$Text_Example->(SetPageNumber: 2)]
[Var:'Text4'=(PDF_Text:'Lasso Professional 7 is server-side software that adds a suite
of dynamic functionality and administration to your Web server. This functionality
empowers you to build and serve just about any dynamic Web application that can
be built with maximum productivity and ease.\n\n', -Type='Paragraph', -Leading=15,
-Font=$Font2)]
```

```

[$Text_Example->(Add: $Text4)]
[Var:'Text5'=(PDF_Text:'Lasso works by using a simple tag-based markup language
(LDML), which can be embedded in Web pages and scripts residing on your Web
server. The details of scripting and programming in LDML are covered in the Lasso
6 Language Guide also included with this product. By default, Lasso Professional 7
is designed to run on the most prevalent modern Web server platforms with the most
popular Web serving applications. In addition, Lasso's extensibility allows Web Server
Connectors to be authored for any Web server for which default connectivity is not
provided.\n\n', -Type='Paragraph', -Leading=15, -Font=$Font2)]
[$Text_Example->(Add: $Text5)]
[$Text_Example->(DrawText: (String: $Text_Example->GetPageNumber),
-Font=$Font2, -Top=30, -Left=560)]
[$Text_Example->Close]

```

PDF Form Example

The following example creates a PDF file that contains both text and a form.

```

<?LassoScript
Var: 'Form_Example' = (PDF_Doc: -File='Form_Example.pdf', -Format='PDF',
-Size='A4');
Var: 'myFont' = (PDF_Font: -Face='Helvetica', -Size='12');
$Form_Example->(AddText:'This PDF file contains a form. See below.\n',
-Font=$myFont);
$Form_Example->(DrawText: 'Select List', -Font=$myFont, -Left=90, -Top=116);
$Form_Example->(AddSelectList: -Name='mySelectList', -Values=(Array: 'one',
'two', 'three', 'four'), -Default='one', -Left=216, -Top=104, -Width=144, -Height=72,
-Font=$myFont);
$Form_Example->(DrawText: 'Pull-Down Menu', -Font=$myFont, -Left=90, -Top=200);
$Form_Example->(AddComboBox: -Name='myComboBox', -Values=(Array: 'one',
'two', 'three', 'four'), -Default='one', -Left=216, -Top=188, -Width=144, -Height=18,
-Font=$myFont);
$Form_Example->(DrawText: 'Text Area', -Font=$myFont, -Left=90, -Top=238);
$Form_Example->(AddTextArea: -Name='myTextArea', -Value='Some text', -Left=216,
-Top=230, -Width=144, -Height=72, -Font=$myFont);
$Form_Example->(DrawText: 'Password Field', -Font=$myFont, -Left=90, -Top=334);
$Form_Example->(AddPasswordField: -Name='myPassword', -Value='****', -Left=216,
-Top=322, -Width=144, -Height=18, -Font=$myFont);
$Form_Example->(DrawText: 'Text Field', -Font=$myFont, -Left=90, -Top=368);
$Form_Example->(AddTextField: -Name='myTextField', -Value='Some More Text',
-Left=216, -Top=360, -Width=144, -Height=18, -Font=$myFont);
$Form_Example->(AddHiddenField: -Name='myHiddenField', -Value='Shh');
$Form_Example->(AddSubmitButton: -URL='http://www.example.com/response.lasso',
-Name='myButton', -Value='Submit', -Caption='Submit Form', -Left=216, -Top=400,
-Width=100, -Height=26, -Font=$myFont);
$Form_Example->(AddResetButton: -Name='Reset', -Value='Reset',

```

```

-Caption='Reset Form', -Left=365, -Top=400, -Width=100, -Height=26,
-Font=$myFont);
$Form_Example->Close;
?>

```

PDF Table Example

The following example creates a PDF file that contains both text and a table.

```

[Var:'Table_Example'=(PDF_Doc: -File='Table_Example.pdf',
                               -Size='A4')]
[Var:'Font1'=(PDF_Font: -Face='Helvetica',
                       -Size='24')]
[Var:'Font2'=(PDF_Font: -Face='Helvetica',
                       -Size='12')]
[Var:'Text'=(PDF_Text:'This PDF file contains a table. See below.\n\n',
                 -Leading=15,
                 -Font=$Font1)]
[Var:'Cell1'=(PDF_Text:'Cell One', -Font=$Font2)]
[Var:'Cell2'=(PDF_Text:'Cell Two', -Font=$Font2)]
[Var:'Cell3'=(PDF_Text:'Cell Three', -Font=$Font2)]
[Var:'Cell4'=(PDF_Text:'Cell Four', -Font=$Font2)]
[$Table_Example->(Add: $Text)]
[Var:'MyTable'=(PDF_Table: -Cols=2,
                           -Rows=2,
                           -Spacing=4,
                           -Padding=4,
                           -Width=75,
                           -BorderWidth=7)]
[$MyTable->(Add: $Cell1, -Col=0, -Row=0,
              -Colspan=1, -Rowspan=1,
              -VerticalAlignment='Center',
              -HorizontalAlignment='Center',
              -BorderWidth=4)]
[$MyTable->(Add: $Cell2, -Col=0, -Row=1,
              -Colspan=1, -Rowspan=1,
              -VerticalAlignment='Center',
              -HorizontalAlignment='Center',
              -BorderWidth=4)]
[$MyTable->(Add: $Cell3, -Col=1, -Row=0,
              -Colspan=1, -Rowspan=1,
              -VerticalAlignment='Center',
              -HorizontalAlignment='Center',
              -BorderWidth=4)]
[$MyTable->(Add: $Cell4, -Col=1, -Row=1,
              -Colspan=1, -Rowspan=1,
              -VerticalAlignment='Center',

```

```

-HorizontalAlignment='Center',
-BorderWidth=4]]
[$Table_Example->(Add: $MyTable)]
[$Table_Example->Close]

```

PDF Graphics Example

The following example shows how to create a PDF file that contains drawn graphic objects.

```

[Var:'Graphic_Example'=(PDF_Doc: -File='Graphic_Example.pdf',
-Height='650',
-Width='550')]
[Var:'Text'=(PDF_Text:'This PDF file contains lines and circles. See below.\n')]
[$Graphic_Example->(Add: $Text)]
[$Graphic_Example->(Line: 200, 400, 400, 400)]
[$Graphic_Example->(Line: 200, 500, 400, 500)]
[$Graphic_Example->(Line: 266, 333, 266, 566)]
[$Graphic_Example->(Line: 333, 333, 333, 566)]
[$Graphic_Example->(Line: 200, 333, 400, 566)]
[$Graphic_Example->(Circle: 233, 366, 20)]
[$Graphic_Example->(Circle: 300, 452, 20)]
[$Graphic_Example->(Circle: 366, 533, 20)]
[$Graphic_Example->(Line: 220, 432, 240, 472)]
[$Graphic_Example->(Line: 220, 472, 240, 432)]
[$Graphic_Example->(Line: 360, 432, 380, 472)]
[$Graphic_Example->(Line: 360, 472, 380, 432)]
[$Graphic_Example->(Line: 220, 517, 240, 558)]
[$Graphic_Example->(Line: 220, 558, 240, 517)]
[$Graphic_Example->Close]

```

PDF Barcode Example

The following example shows how to create a PDF file that contains text accompanied by a barcode.

```

[Var:'Barcode_Example'=(PDF_Doc: -File='Barcode_Example.pdf',
-Height=172,
-Width=300)]
[Var:'Font1'=(PDF_Font: -Face='Courier', -Size='12')]
[Var:'MyBarcode'=(PDF_Barcode: -Type='CODE39',
-Code='1234567890',
-GenerateChecksum,
-ShowCode39StartStop,
-TextSize: 6)]

```



```
[$Barcode_Example->(DrawText: 'The Shipping Company\n',
                        -Font=$Font1, -Left=72, -Top=90)]
[$Barcode_Example->(Add: $MyBarcode, -Left=72, -Top=40)]
[$Barcode_Example->Close]
```

Serving PDF Files

This section describes how PDF files can be served using Lasso Professional 7. This can be done by supplying a download link to the created PDF file, or by using the [PDF_Serve] tag described in this chapter.

Syntax Note: When creating PDF files using LDML and serving data to a browser in the same page, the use of the LassoScript syntax is recommended as it does not output hard returns in the rendered HTML source code. For more information on LassoScript, see *Chapter 24: LassoScript*.

Linking to PDF Files

Named PDF files may be linked to in a format file using basic HTML. Once a user clicks on a link to a file with a .pdf extension, the client browser should prompt to download the file or launch the file in PDF reader (if configured to do so).

To link a PDF file:

The example below shows how a PDF can be created and written to file, and then linked to in the format file.

```
<?LassoScript
  Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',
                      -Size='A4');
  Var:'Text'=(PDF_Text: 'Hello World');
  $MyFile->(Add: $Text);
  $MyFile->Close;
?>
<html>
<body>
<p>Click on the following link to download MyFile.pdf.</p>
<p><a href="MyFile.pdf">Click Here</a></p>
</body>
</html>
```

Serving PDF Files to Client Browsers

PDF files may also be served directly to a client browser using the [PDF_Serve] tag. This tag automatically informs the client Web browser that the data being load is a PDF file, and outputs the file with the correct file name. If the client Web browser is configured to handle PDF files via a reader, then the out PDF file will automatically be opened in the clients configured PDF reader. Otherwise, the client Web browser should prompt the user to save the file.

Table 23: PDF Serving Tags

Tag	Description
[PDF_Serve]	Serves a PDF file to a client browser with a MIME type of application/pdf. Requires a -File parameters which specifies the name of the file to be output to the browser. An optional -Type parameter may be used to specify additional MIME types.

To serve a PDF file to a client browser:

Use the [PDF_Serve] tag to serve the created PDF file. The file parameter specifies the file name that should be output.

```
<?LassoScript
  Var:'MyFile'=(PDF_Doc: -File='MyFile.pdf',
                    -Size='A4',
                    -NoCompress);
  Var:'Text'=(PDF_Text: 'Hello World');
  $MyFile->(Add: $Text);
  $MyFile->Close;
  PDF_Serve: $MyFile, -File='MyFile.PDF'
?>
```

To serve a PDF file without writing to file:

PDF files may be served to the client browser without ever writing them to file on the local server. This is done using the [PDF_Doc] tag without the -File parameter. This allows a PDF file to be created in the system memory, but does not the save the file to a hard drive location. The resulting file can saved by the end user to a location on the end user's hard drive.

```
<?LassoScript
  Var:'MyFile'=(PDF_Doc: -Size='A4',
                    -NoCompress);
```

```
Var:'Text'=(PDF_Text: 'Hello World');  
$MyFile->(Add: $Text);  
$MyFile->Close;  
PDF_Serve: $MyFile, -File='MyFile.pdf';  
?>
```




Section V

Upgrading

This section contains detailed instructions for developers who are upgrading solutions developed for Lasso Web Data Engine 3.x to Lasso Professional 7.

- *Chapter 31: Upgrading Your Solutions* includes complete instructions for upgrading solutions that were built using Lasso Professional 5 or Lasso Web Data Engine 3.x for compatibility with Lasso Professional 7.

This section should be read in concert with *Chapter 13: Upgrading* in the Lasso Professional 7 Setup Guide.

31

Chapter 31

Upgrading Your Solutions

This chapter contains important information for users of Lasso Professional 6, Lasso Professional 5, or Lasso Web Data Engine 3.x who are upgrading to Lasso Professional 7. Please read through this chapter before attempting to run solutions in Lasso Professional 7 that were originally developed for an earlier version of Lasso.

This chapter is split into three main sections. The first covers changes for those upgrading from Lasso Professional 6. The next covers changes for those upgrading from Lasso Professional 5. The remainder of the chapter covers changes for those upgrading from Lasso WDE 3.x.

Topics in this chapter include:

- **Introduction** includes general information about what has changed in Lasso Professional 7.
- **Unicode Support** describes how Lasso uses Unicode internally and how Lasso translated to and from other character encodings automatically.
- **Bytes Type** describes the new bytes type in Lasso Professional 7, lists the tags that return data in the bytes type, and compares the bytes type to the string type.
- **Syntax Changes (Lasso 6)** contains information about what LDML syntax constructs have changed since Lasso Professional 6.
- **Tag Name Changes (Lasso 5/6)** details the tag names which have been changed in LDML 7 since LDML 6.
- **Syntax Changes (Lasso 5)** contains information about what LDML syntax constructs have changed since Lasso Professional 5.
- **Lasso MySQL (Lasso 5)** contains information about changes made to the Lasso Connector for Lasso MySQL since Lasso Professional 5..

- *Syntax Changes (Lasso WDE 3.x)* contains information about what LDML syntax constructs have changed since Lasso WDE 3.x and how to update format files which use those syntax constructs.
- *Tag Name Changes (Lasso WDE 3.x)* details the tag names which have been changed in LDML 7 since LDML 3.
- *Unsupported Tags (Lasso WDE 3.x)* lists the LDML 3 tags that are no longer supported in Lasso Professional 7.
- *FileMaker Pro (Lasso WDE 3.x)* contains information about how to update a solution which depended on the Apple Event based FileMaker Pro data source module to the new Lasso Connector for FileMaker Pro.

This chapter does not attempt to cover every issue that users of versions of Lasso *prior* to Lasso Web Data Engine 3.x may encounter.

Introduction

Blue World has strived to make this upgrade as painless as possible for existing customers. All Lasso Professional 6 sites which were written using strict and preferred syntax should run without modifications in Lasso Professional 7 except for the issues mentioned in this chapter. Lasso Professional 5 and Lasso WDE 3.x sites generally require some updates in order to work in Lasso Professional 7.

The goals for improvements to the Lasso product family with Lasso Professional 7 were to:

- Improve the speed of Lasso by introducing a new parser and internal byte code interpreter.
- Provide Unicode support throughout Lasso. All string manipulations are now performed using 16-bit Unicode encoding. All string output is in the UTF-8 character set.
- Streamline database communication by re-using the same connection to the data source. New tags make manipulating database results even easier.
- Introduce objects for file operations, image operations, and network operations.
- Introduce new syntax shortcuts that make writing LDML code easier. New tags perform common tasks with less code.
- Extend custom tags and data types with new features. Many limitations of custom tags have been removed.

Significant effort has been expended to ensure that existing solutions will continue to run in Lasso Professional 7 with few if any modifications

required. However, please read through this chapter to learn about changes that may require modifications to your solutions.

Lasso Studio and LDML Updater

Lasso Studio includes an LDML Updater that can be used on code from earlier versions of Lasso to bring it into compliance with the latest version of LDML. See the documentation for Lasso Studio for more information.

Unicode Support

Lasso Professional 7 introduces Unicode support throughout Lasso Service, the database connectors, and LDML. This is a significant architectural change that alters how all string and binary data is processed by Lasso.

The Unicode standard defines a universal character set which includes characters for just about every language on the planet. The transition to a full Unicode workflow should make it easier to transfer data that contains characters which formerly required special-purpose encodings between different applications.

Unicode is rapidly achieving dominance as the standard encoding for data on the Internet, in leading operating systems, and in database products. Recently, Mac OS X and Windows have both implemented native support for Unicode. All current leading Web browsers support Unicode data. Many text editors such as BareBones BBEdit have recently introduced native support for Unicode. And, future database offerings from MySQL and other database vendors are expected to offer full support for Unicode encoding.

Every effort has been made to make the change to Unicode transparent to the Lasso developer. However, these architectural changes may require modification of some Lasso Professional 6 sites and may require some additional planning and coding in order to work with Web browsers and databases that do not yet support Unicode.

The following list includes details about how Unicode is supported in Lasso Professional 7 and also includes details about backwards compatibility.

Note: Please also read the following section on the new *Bytes Type* for details about how binary and string data is handled using LDML tags.

- **Format Files** – If a format file contains a valid byte-order mark it is read using the UTF-8 character encoding. If no byte-order mark is read then the format file is assumed to be encoded in the Macintosh (or

Mac-Roman) character set on Mac OS X or the Latin-1 character set (also known as ISO 8859-1) on Windows or Linux.

Popular text editors such as BBEdit can encode text files using UTF-8 and will insert the proper byte-order mark that Lasso needs to identify the character set of the format file. Consult the documentation for the text editor for more information.

All existing format files will be read using the Macintosh or Latin-1 (ISO 8859-1) character sets which Lasso Professional 6 used. New format files which need to take advantage of the extended character set that Unicode offers should be encoded as UTF-8 and include a proper byte-order mark.

Note: Lasso does not support format files encoded using the UTF-16 or UTF-32 character sets.

- **Web Browser** – By default all files sent to the Web browser by Lasso Professional 7 will be encoded using UTF-8. The default page encoding option in the *Settings > Global > Syntax* section of Lasso Administration can be used to change the default encoding to the Lasso 6 (pre-Unicode) standard Latin-1 character set (also known as ISO 8859-1).

If encoding different from the default is needed for a given format file the [Content_Type] tag can be used to instruct Lasso to encode the returned page using a different character set. For example the following tag instructs Lasso to use the Latin-1 (ISO 8859-1) character set.

```
[Content_Type: 'text/html;charset=iso-8859-1']
```

The [Content_Type] tag sets the page variable `__encoding__` to the desired charset. Modifying this variable will also change the character set that will be used to return the page to the client's Web browser.

- **Forms** – Most Web browsers return data from HTML forms using the same encoding that was used to transmit the Web page to them. Lasso assumes that all incoming form data is going to use the default page encoding which is set in Lasso Administration in the *Settings > Global > Syntax* section. This means that all incoming form data must use either UTF-8 or Latin-1 (ISO 8859-1) encoding.

It is recommended to use UTF-8 as the default page encoding since this is the emerging Internet standard. However, if forms are being submitted to Lasso from Web pages that do not use UTF-8 (or from pre-Unicode browsers) it may be necessary to change the default page encoding so data in the forms will be interpreted properly.

- **Database Connector** – Lasso communicates with each database using the character set specified in the table settings in Lasso Administration.

The character set for MySQL databases can be set to either UTF-8 or Latin-1 (ISO 8859-1).

By default, the Lasso Connectors for MySQL communicate with existing databases using the Latin-1 (ISO 8859-1) character set. If desired, the character set for each table can be changed to UTF-8 in the *Setup > Data Sources > Tables* section of Lasso Administration.

SQL statements sent using the `-SQL` parameter are encoded similarly. If the `-Table` parameter is specified then the character set for that table will be used. If no `-Table` parameter is specified then the SQL statement will be encoded using the Latin-1 (ISO 8859-1) character set.

The Lasso Connector for FileMaker Pro uses Latin-1 (ISO 8859-1) encoding by default on Windows and Linux. Macintosh (or Mac-Roman) encoding is used by default on Mac OS X. If required, the character set for each database can be changed in the *Setup > Data Sources > Databases* section of Lasso Administration.

The Lasso Connector for JDBC always uses UTF-8 on any platform.

Since the default character set encoding for each database connector is the same as that used in Lasso Professional 6, no changes should be required to existing solutions. However, any database containing extended characters must continue to use the same character encoding or stored data may not be interpreted properly when it is retrieved from the database.

- **LDML Tags** – All LDML tags process string data as double-byte Unicode strings. Character encoding is only performed when data is imported into Lasso or exported according to the rules specified above. The bytes type can be used to process binary data and to perform low-level character set translation if required.

See the following section for details of what LDML tags return data in the bytes type and how it compares with the string type.

Bytes Type

Since all string data is now processed using double-byte Unicode strings it is necessary to introduce a new data type that stores single-byte binary data strings. This new data type in Lasso Professional is called the bytes type and is manipulated using the `[Bytes]` tag and associated member tags. Data in the bytes type is often referred to as a byte stream.

The bytes type adds two important abilities to Lasso Professional 7. Binary data can be treated separately from string data and data can be converted between single-byte character sets directly within Lasso. The bytes type is

fully documented in the Extending Lasso Guide. Please see that manual for additional details about the bytes type and the member tags that it supports.

The bytes type is used to return any strings in Lasso that will potentially contain binary data. Many substitution tags always return byte streams or do so under certain circumstances. These tags are listed in the following *Table 1: Tags That Return the Bytes Type*.

Table 1: Tags That Return the Bytes Type

Tag	Description
[Bytes]	Used to create a new bytes buffer or to cast a string to the bytes type. Many of the member tags of the bytes type also return byte streams.
[Decompress]	Always returns a byte stream.
[Decrypt_BlowFish]	Always returns a byte stream.
[Encode_Base64]	Always returns a byte stream.
[Encrypt_BlowFish]	Always returns a byte stream.
[Field]	Returns a byte stream only for MySQL fields of type BLOB.
[Field_Read]	Always returns a byte stream.
[File_ReadLine]	Always returns a byte stream.
[File->Read]	Always returns a byte stream.
[Include_Raw]	Always returns a byte stream.
[Include_URL]	Always returns a byte stream.
[Net->Read]	Always returns a byte stream.
[Net->ReadFrom]	Always returns a byte stream.
[String_ReplaceRegExp]	Returns a byte stream if the input is a byte stream, otherwise returns a string.
Other Tags	Many tags in LDML such as [Array->Get] or [Map->Find] return data in the same type it was stored. These tags may also return data in the bytes type.

Bytes and Strings

The bytes type and the string type support many of the same member tags. These member tags can be used on either byte streams or strings without worrying about the underlying data type. The shared member tags are listed in *Table 2: Byte and String Shared Member Tags*.

In addition to these tags both the bytes type and the string type support the standard comparison operators ==, !=, >>, !>>, ===, and !==. The bytes

type also supports the + and += symbol for appending data to the end of the stream.

Table 2: Byte and String Shared Member Tags

Tag	Description
[Bytes->Append]	Appends the specified characters onto the end of the byte stream.
[Bytes->BeginsWith]	Returns true if the byte stream begins with the specified characters. Case sensitive.
[Bytes->Contains]	Returns true if the byte stream contains the specified characters. Case sensitive.
[Bytes->EndsWith]	Returns true if the byte stream ends with the specified characters. Case sensitive.
[Bytes->Find]	Returns the position of the specified characters within the byte stream. Case sensitive.
[Bytes->Get]	Returns a specified character from the byte stream.
[Bytes->Length]	Returns the length of the byte stream in bytes.
[Bytes->RemoveLeading]	Removes the specified characters from the beginning of the byte stream. Case sensitive.
[Bytes->RemoveTrailing]	Removes the specified characters from the end of the byte stream. Case sensitive.
[Bytes->Replace]	Replaces the specified characters in the byte stream with the specified replacement. Case sensitive.
[Bytes->Size]	Returns the length of the byte stream in bytes.
[Bytes->Split]	Splits the byte stream on the specified character. Case sensitive.
[Bytes->Trim]	Trims ASCII white space characters from the start and end of the byte stream. Removes spaces, tabs, return characters, and newline characters.

The table above includes all of the most commonly used string member tags. These tags help to make the string and bytes types generally interchangeable. However, there are a significant number of string member tags that are not supported by the bytes type. These are listed in **Table 3: Unsupported String Member Tags**.

In order to use any of these member tags on byte streams the data must first be converted to a string. Information on how to convert data to and from the bytes type and string type is included in the next section.

Table 3: Unsupported String Member Tags

Tag	Description
Character Tags	Tags which fetch information about the characters in a string including [String->CharDigitValue], [String->CharName], and [String->CharType].
Comparison Tags	Tags which compare strings with case sensitivity. [String->Compare] and [String->CompareCodePointOrder]
Case Tags	Tags which report the case of a string including [String->FoldCase], [String->LowerCase], [String->TitleCase], and [String->UpperCase].
Case Modification Tags	Tags which change the case of a string including [String->ToLower], [String->ToTitle], and [String->ToUpper].
Character Is Tags	Tags which report information about the characters within a string including all tags starting with [String->Is...].
Miscellaneous Tags	The following tags are also not supported by the bytes type: [String->Digit], [String->Merge], [String->PadLeading], [String->PadTrailing], [String->Remove], [String->Reverse], [String->Substring], and [String->Unescape].

Converting From Bytes to Strings

Data can be converted from byte streams to strings easily, but the method differs depending on what character set the byte stream is encoded in and how the data is going to be used.

- **Automatic Casting** – When a byte stream is passed to a substitution or process tag that is expecting a string value it is automatically cast to type string. For example, the [String_...] substitution tags automatically cast their parameters to strings.
- **Explicit Casting** – The [String] tag can be used to explicitly cast a byte stream to a string. The byte stream will be converted by assuming it is encoded using the UTF-8 character set. Explicit casting is appropriate for data read in using the [Include_URL] or [Net->Read] tags since most communication on the Internet is encoded using this character set.
- **Converting Character Sets** – The [Bytes->ExportString] tag can be used to convert a byte stream that is encoded using a character set other than UTF-8 into a string. The tag accepts a single parameter which specifies what character set the byte stream is encoded in and returns a string (encoded in the default Unicode double-byte encoding that Lasso uses internally for all strings).

For example, a file can be read in the Mac-Roman character set and converted to a string using this code.

```
[Var: 'myfile' = (File_Read: 'myfile.txt')]
[Var: 'myststring' = $myfile->(ExportString: 'Mac-Roman')]
```

Similar methods can be used to convert strings into byte streams. Tags that expect a byte stream as a parameter automatically cast strings to byte streams. These tags include [File_Write], [File_WriteLine], [File->Write], [Net->Write], etc. The [Bytes] tag explicitly casts strings to a byte stream. The [Bytes-ImportString] tag with a string parameter and an encoding parameter can be used to import a string into a byte stream converting it to any desired character encoding.

Bytes Member Tags

The bytes type supports a number of additional member tags which are documented in full in the Extending Lasso Guide. Please see that manual for more information.

Updating Existing Sites

In order to promote backwards compatibility the bytes type supports the core member tags of the string type and Lasso performs automatic conversions between the two types when necessary.

However, there are a couple situations which will require Lasso Professional 6 sites to be updated in order to work properly with Lasso Professional 7. These situations are detailed below.

- **Checking for String Type** – Byte streams are of type bytes so explicit checks for type string will fail. For example, the following code reads a file into a variable and then checks the type of the variable.

```
[Var: 'myfile' = (File_Read: 'myfile.txt')]
[If: $myfile->type == 'string']
...
[/If]
```

The conditional will fail since the variable myfile is of type bytes rather than type string. The conditional can be changed to the following to create code that works in either Lasso Professional 6 or 7.

```
[Var: 'myfile' = (File_Read: 'myfile.txt')]
[If: ($myfile->type == 'string') || ($myfile->type == 'bytes')]
...
[/If]
```

- **String Member Tags** – If any string member tags are used on a byte stream which are not supported by the bytes type then an error will

occur. For example, this code to read in a file and then convert it to uppercase will fail in Lasso Professional 7 since the tag [String->toUpper] is not implemented for the bytes type.

```
[Var: 'myfile' = (File_Read: 'myfile.txt')]
[$myfile->toUpper]
```

There are two solutions to this issue. The easiest is to cast the output of [File_Read] to a string before storing it in a variable. This solution can be applied across a site by doing a search for each of the tags that return byte streams and adding an explicit cast using the [String] tag.

```
[Var: 'myfile' = (String: (File_Read: 'myfile.txt'))]
[$myfile->toUpper]
```

Another possibility is to use a substitution tag rather than a member tag to perform the string conversion. The substitution tag will automatically cast the byte stream to a string and will return a string value.

```
[Var: 'myfile' = (File_Read: 'myfile.txt')]
[Var: 'myfile' = (String_UpperCase: $myfile)]
```

Syntax Changes (Lasso 6)

Lasso Professional 7 introduces changes to some of the core syntax rules of LDML. Most of these changes were made to improve the reliability and error reporting of Lasso Professional 7. Some of these changes may require you to rewrite portions of your existing Lasso-based solutions for full compatibility with Lasso Professional 7. This section describes each change, why it was made and how to update existing format files.

Table 4: Syntax Changes

Syntax Change	Description
Strict Syntax	Strict syntax is now required: all parameter keywords must be preceded by a hyphen, all string literals must be surrounded by quote marks, and all tag names must be defined before being called.
Recursion Limit	A limit can be configured on the depth of nested [Include] and [Library] tags allowed. By default the limit is a depth of 50.
Format File Execution Time Limit	A limit can be configured on the maximum amount of time that a format file will be allowed to execute. By default the limit is 10 minutes.
Internal Tags	Many tags are now implemented as part of the LDML parser in order to prove better performance.

Iterate Enhancement	The [Loop_Count] and [Loop_Abort] tags can now be used within [Iterate] ... [/Iterate] tags.
Custom Tags Enhancement	Database results can now be retrieved from within custom tags.
Miscellaneous Shortcuts	A number of syntax shortcuts have been introduced. See the full description below for details.
Unicode Support	All strings are now processed using double-byte Unicode and output in UTF-8 format by default. See also the discussions of Unicode Support and the Bytes Type which precede this section.
Classic Lasso	Classic Lasso support is disabled by default and its use has been deprecated. Solutions relying on Classic Lasso should be transition to Inline-based methodology.
-Email... Command Tags	The -Email... commands are no longer supported in LDML 7. The [Email_Send] tag must be used instead.
Decimal Precision	Decimal numbers are output using the fewest number of significant digits possible.
Member Tags and Parentheses	Member tags which have multiple parameters must be surrounded by parentheses.
PDF -Top Parameter	The -Top parameter in various PDF tags always measures from the top margin of a document.
Global Variables	Use the [Global] tag rather than the [Variable] tag to reference global variables.
[NSLookup]	Due to changes in Mac OS X 10.3 reverse lookups may not work with all DNS servers.
[Repetition] Tag	The [Repetition] tag has been deprecated. Rewriting pages to use the modulus symbol % will result in better performance.
[TCP_...] Tags	The [TCP_...] tags have been deprecated in favor of the new [Net] type and its member tags.
[Else:If] and [Else_If]	These tags are no longer supported. Use [Else] instead.
[LoopCount] and [LoopAbort]	These tags are no longer supported. Use [Loop_Count] and [Loop_Abort] instead.

Container Tags	Container tags must be defined within LassoStartup. Container tags cannot be defined on-the-fly. New keywords allow both looping and simple container tags to be created.
Custom Tags	Parameters and return values are now passed by reference. [PreCondition] and [PostCondition] are no longer supported. Asynchronous tag update.
XML Tags	The XML tags have been re-implemented. Some modifications to existing sites may be required.
[Encode_ISOtoMac]	This tag and [Encode_MacToISO] have been deleted. Their functionality can be replicated using the new [Bytes] type.

Strict Syntax

Lasso Professional 6 introduced the option to use strict syntax checking. This option was on by default, but could be turned off for better backwards compatibility with Lasso Professional 5 and earlier.

In Lasso Professional 7, strict syntax checking is now required. It can no longer be deactivated.

With strict syntax the following rules are enforced:

- All keyword parameters to built-in and custom tags must include a hyphen. This helps to find unknown tag parameters and to catch other common syntax errors.
- All string literals must be surrounded by quotes. This helps to prevent accidental calls to tags, to identify undefined variables, and to catch other common syntax errors.
- All tag calls must be defined. Unknown tags will no longer simply return the tag value as a string.

With strict syntax any of the errors above will be reported when a page is first loaded. They must be corrected before the code on the page will be executed. When upgrading to Lasso Professional 7 it is advisable to first try existing Lasso Professional 5 and 6 sites and correct any errors that are reported.

To update existing sites for strict syntax:

If a site is relatively small then the easiest method is to load each Web page and see if any errors are reported. The following tips can be used for a more methodical search.

- Check that all string literals are surrounded by quotes. Quotes are not necessary around integers or decimal numbers, hyphenated keyword

parameters, tag names, or variable names when used with the `&` or `#` symbols.

- Check that all keywords in tag calls are preceded by a hyphen. Keyword and keyword/value parameters must be preceded by a hyphen, but do not need to be quoted. Name/value parameters should include quotes around both the name and value (unless they are numbers).
- Check that all command tags used within opening `[Inline]` tags are preceded by a hyphen. Quotes are not necessary around command tags, even when they are specified within an array.
- Check that all client-side JavaScript is formatted properly. JavaScript should either be included in `[NoProcess] ... [NoProcess]` tags or HTML comment tags `<!-- ... -->` which ensure that no LDML code within is processed. Or, any square brackets which are required within the JavaScript should be output from an `[Output]` tag.

`[Output: 'array[4]]'`

→ `[array[4]]`

Recursion Limit

Lasso includes a limit on the depth of recursive include files. This limit can help prevent errors or crashes caused by some common coding mistakes. The limit sets the maximum depth of nested `[Include]` and `[Library]` tags that can be used. If the depth is exceeded then a critical error is returned and logged.

The recursion limit is set to 50 by default and can be modified or turned off in the *Setup > Global > Settings* section of Lasso Admin.

Note that the limit does not apply to the number of `[Include]` and `[Library]` tags within a single file, but to the depth reached using an `[Include]` tag to include a file that itself uses an `[Include]` tag to include another file and so on.

Format File Execution Time Limit

Lasso includes a limit on the length of time that a format file will be allowed to execute. This limit can help prevent errors or crashes caused by infinite loops or other common coding mistakes. If a format file runs for longer than the time limit then it is killed and a critical error is returned and logged.

The execution time limit is set to 10 minutes (600 seconds) by default and can be modified or turned off in the *Setup > Global > Settings* section of Lasso Admin. The execution time limit cannot be set below 60 seconds.

The limit can be overridden on a case by case basis by including the `[Lasso_ExecutionTimeLimit]` tag at the top of a format file. This tag can set the time limit higher or lower for the current page allowing it to exceed the default time limit. Using `[Lasso_ExecutionTimeLimit: 0]` will deactivate the time limit for the current format file altogether.

On servers where the time limit should be strictly enforced, access to the `[Lasso_ExecutionTimeLimit]` tag can be restricted in the **Setup > Global > Tags** and **Security > Groups > Tags** sections of Lasso Admin.

Asynchronous tags and compound expressions are not affected by the execution time limit. These processes run in a separate thread from the main format file execution. If a time limit is desired in an asynchronous tag the `[Lasso_ExecutionTimeLimit]` tag can be used to set one.

Note: When the execution time limit is exceeded the thread that is processing the current format file will be killed. If there are any outstanding database requests or network connections open there is a potential for some memory to be leaked. The offending page should be reprogrammed to run faster or exempted from the time limit using `[Lasso_ExecutionTimeLimit: 0]`. Restarting Lasso Service will reclaim any lost memory.

Internal Tags

Many LDML tags are now implemented directly in the LDML parser in order to provide better performance. Since the new versions of these tags implement the same functionality as the old version of these tags no changes to existing solutions are required. However, the `[Lasso_TagExists]` tag will report `False` for all of the internal tags.

The internal tags include:

```
[Abort], [Define_Tag] ... [/Define_Tag], [Define_Type] ... [/Define_Type],
[Encode_Set] ... [/Encode_Set], [Fail], [Fail_If], [False], [Handle] ... [/Handle],
[Handle_Error] ... [/Handle_Error], [If] ... [Else] ... [If], [Iterate] ... [/Iterate],
[Lasso_Abort], [Loop] ... [/Loop], [Loop_Abort], [Loop_Count], [NoProcess], [Params],
[Protect] ... [/Protect], [Return], [Run_Children], [Select] ... [Case] ... [/Select], [Self],
[True], and [While] ... [While].
```

Iterate Enhancement

In Lasso Professional 6 the `[Iterate] ... [/Iterate]` tags did not support the use of the `[Loop_Count]` or `[Loop_Abort]` tags. These tags have been rewritten in Lasso Professional 7 so that all looping container tags now function identically.

In the following example the `[Loop_Count]` is output on each iteration and the iteration is stopped after the item `Beta` is seen.

```
[Iterate: (Array: 'Alpha', 'Beta', 'Gamma'), (Var: 'Temp')]
  <br>[Loop_Count]: [Var: 'Temp']
  [If: $Temp == 'Beta']
    [Loop_Abort]
  [/If]
[/Iterate]
```

→
1: Alpha

2: Beta

For more information about the [Iterate] ... [/Iterate] tags see *Chapter 12: Conditional Logic*.

Custom Tags Enhancement

In Lasso Professional 6 it was not possible to get to the results of a database action from within a custom tag. In Lasso Professional 7 this limitation has been removed. It is now possible to write custom tags which work directly with [Field] data, the [Found_Count], [Action_Params], or any other values.

As a demonstration of this new ability the [Link_...] tags have all been rewritten as custom tags.

In the following example, a custom tag returns a string describing the results of a database action.

```
[Define_Tag: 'Ex_Results']
  [Return: 'Showing ' + (Shown_Count) + ' records of ' + (Found_Count) + ' found.']
[/Define_Tag]
```

This tag can be used as follows.

```
[Inline: -Findall, -Database='Contacts', -Table='People', -MaxRecords=4]
  [Ex_Results]
[/inline]
```

→ Showing 4 records out of 8 found.

For more information about the [Define_Tag] tag and custom tags see *Chapter 3: Custom Tags* in the Extending Lasso Guide.

Miscellaneous Shortcuts

A number of shortcuts have been introduced in Lasso Professional 7 which will make coding Web sites even easier. There is no need to use any of these shortcuts. The equivalent syntax from earlier versions of Lasso will work fine.

- **Not Contains Symbol** – The negation of the contains symbol >> is now available as !>>. This makes it easy to check that a substring is not

contained in a given string. The following example confirms that Green is not a part of Blue World.

```
[Output: ('Blue World' !>> 'Green')]
```

→ True

- **Equivalence Symbol** – The equals symbol == checks that two values are equal by casting them to the same data type. The new equivalence symbol === checks that two values are equal in both value and data type. The following example shows four expressions that are True using the equals symbol ==.

```
[Output: ('Alpha' == 'Alpha')] → True
```

```
[Output: ('100' == 100)] → True
```

```
[Output: (3.00 == 3)] → True
```

```
[Output: (True == 1)] → True
```

When the equivalency symbol === is used instead only the first expression is True. The rest of the expressions are False since the data types of the two operands are different. The second expression compares a string to an integer. The third expression compares a decimal to an integer. And, the fourth expression compares a boolean to an integer.

```
[Output: ('Alpha' === 'Alpha')] → True
```

```
[Output: ('100' === 100)] → False
```

```
[Output: (3.00 === 3)] → False
```

```
[Output: (True === 1)] → False
```

- **String Concatenation** – Strings are now concatenated together without using the + symbol. In the following example database results are formatted without using the + symbol.

```
[Output: 'Showing ' (Shown_Count) ' records of ' (Found_Count) ' found.']
```

→ Showing 4 records out of 8 found.

- **Array Creation** – The : symbol can be used for array creation. Basically Array: is equivalent to simply ∴.

```
[: 'Alpha', 'Beta', 'Gamma']
```

→ (Array: 'Alpha', 'Beta', 'Gamma')

- **Tag References** – The \ symbol can be used to reference a tag object based on its name. This allows the descriptions of tags to be fetched or for tags to be called with programmatically defined parameters. The following example shows what the output might be for the [Field] tag.

```
[Var: 'myTag' = \Field]
```

```
<br>[Output: $myTag->Description]
```

```
<br>[Output: $myTag->(Run: -Name='Field', -Params='First_Name')]
```

→ `
`A tag that returns a field value.
`
`John

See *Chapter 5: Advanced Programming Topics* in the Extending Lasso Guide for more information.

Unicode Support

All strings in Lasso Professional 7 are represented internally by double-byte Unicode values. This makes it efficient to work with extended characters in a platform neutral fashion. All output from Lasso, whether to the client's Web browser or into a database, is formatted in UTF-8 by default.

UTF-8 is a Unicode standard that is backwards compatible with common 8-bit ASCII character sets. Any extended Unicode characters are encoded using an entity like `E26;` where 4E26 is a hexadecimal number representing the Unicode value for the character.

Classic Lasso

Classic Lasso refers to the ability of Lasso to interpret command tags which are included in URLs or HTML forms and process the action described by those command tags before a format file is loaded.

In prior versions of Lasso this was the sole means of performing database actions. Since Lasso WDE 3.x it has been possible to perform database operation using the `[Inline] ... [/Inline]` tags instead. It is preferable to use this inline methodology for the following reasons.

- The database, table, and field names which are being accessed need never be revealed to the client.
- It is impossible for clients to create new URLs or HTML forms that perform unintended actions.
- The amount of data passed in URLs to and from the client can be greatly reduced. This can provide easier to read and easier to bookmark URLs.
- `[Inline] ... [/Inline]` tags support a number of advanced features like named inlines and accepting arrays of parameters which make it easier to separate the logic of a Web site from the presentation.
- Some actions such as issuing SQL statements to Lasso MySQL require using `[Inline] ... [/Inline]` functionality already.

Note: It is possible to enable Classic Lasso syntax in the **Setup > Global > Syntax** section of Lasso Administration, however since this functionality has been deprecated it will not be supported in a future version of Lasso. It is recommended that sites be transitioned over to inline methodology when used with Lasso Professional 7.

To update existing sites:

The [Action_Params] tag can be used to pass all parameters from the URL or HTML form that loaded the current page to an opening [Inline] ... [/Inline] tag.

In the result page, surround the part of the page that references database results with [Inline] ... [/Inline] tags. The opening [Inline] tag should have a single parameter of [Action_Params]. Often, the [Inline] ... [/Inline] tags can simply surround the entire page contents.

```
[Inline: (Action_Params)]
... Page Contents and Database Action Results ...
[/Inline]
```

The [Inline] ... [/Inline] tags must not be contained within any other [Inline] ... [/Inline] tags. The [Inline] ... [/Inline] tags must surround all [Records] ... [/Records], [Field], [Found_Count], [Link_...], [Error_CurrentError] and other tags that will return the database results.

In order to enhance security, command tags such as the -Database, -Table, and action can be added as parameters to the opening [Inline] tag. These parameters should be placed after the [Action_Params] parameter and will override any conflicting parameters from the URL or HTML form that loads the result page.

For example, the following [Inline] will always perform a -Search action on the People table of the Contacts database even if a -FindAll or -Delete action is specified in the URL.

```
[Inline: (Action_Params),
-Search
-Database='Contacts',
-Table='People',
-KeyField='ID']
... Page Contents and Database Action Results ...
[/Inline]
```

Now that the -Database, -Table, and action are specified in the opening [Inline] tag they can be removed from the URL or HTML form that loads the response page. Any command tags or name/value parameters which will be specified by the client should be left in the URL or HTML form, but static command tags can be moved as parameters into the opening [Inline] tag.

-Email... Command Tags

The -Email... commands are no longer supported in LDML 7. Enabling Classic Lasso syntax will not enable this functionality. The only way to send email through Lasso Professional 7 is to use the [Email_Send] tag.

To update existing sites:

- If emails are being sent using the [Inline] ... [/Inline] tags they can be modified to use the [Email_Send] tag as follows. The following shows the old approach based on -Email... command tags.

```
[Inline: -Email.Host='mail.example.com',
-Email.To='me@example.com',
-Email.From='me@example.com',
-Email.Subject='An Example Email Message',
-Email.Format='email_format.lasso']
[/Inline]
```

The syntax for [Email_Send] is very similar. Notice that -Email.Format has been changed to -Body=(Include: ...). This is the preferred method of including another format file as the body of an email message.

```
[Email_Send: -Host='mail.example.com',
-To='me@example.com',
-From='me@example.com',
-Subject='An Example Email Message',
-Body=(Include: 'email_format.lasso')]
```

- If the inline performs a database search in addition to sending an email message the two functions must be factored out as follows. The following example performs a search and sends a single email message.

```
[Inline: -FindAll,
-Database='Contacts',
-Table='People',
-Email.Host='mail.example.com',
-Email.To='me@example.com',
-Email.From='me@example.com',
-Email.Subject='An Example Email Message',
-Email.Format='email_format.lasso']
[Records]
...
[/Records]
[/Inline]
```

In the replacement the -Email... tags are removed from the opening [Inline] tag and the [Email_Send] tag is placed within the [Inline] ... [/Inline] tags, but not within the [Records] ... [/Records] tags. If [Email_Send] is placed in the [Records] ... [/Records] tags then one email for each found record will be sent.

```
[Inline: -FindAll,
-Database='Contacts',
-Table='People']
[Email_Send: -Host='mail.example.com',
-To='me@example.com',
```

```

-From='me@exmaple.com',
-Subject='An Example Email Message',
-Body=(Include: 'email_format.lasso')]
[Records]
...
[/Records]
[/Inline]

```

- If emails are being sent using command tags in a URL they can be modified to use the [Email_Send] tag as follows.

```

<a href="default.lasso?-Email.Host=mail.example.com&
-Email.To=me@example.com&-Email.From='me@exmaple.com&
-Email.Subject='An Example Email Message&
-Email.Format=email_format.lasso"> Send Email </a>

```

The URL should be simplified to contain just the name of the format file.

```

<a href="default.lasso"> Send Email </a>

```

The file default.lasso then must be augmented with the [Email_Send] tag. Notice that -Email.Format has been changed to -Body=(Include: ...). This is the preferred method of including another format file as the body of an email message.

```

[Email_Send: -Host='mail.example.com',
-To='me@example.com',
-.From='me@exmaple.com',
-.Subject='An Example Email Message',
-Body=(Include: 'email_format.lasso')]

```

The same technique can be used to modify an HTML form. Simply remove the -Email... command tags from the form and place an [Email_Send] tag on the response file.

Decimal Precision

Decimal numbers are output using the fewest number of significant digits required. In prior versions of Lasso decimal numbers were always output by default using six significant digits. For example, the following math calculation outputs only two significant digits.

```

[Output: 2.02 + 2.0400] → 4.06

```

In general the output from LP7 should be more readable than the output from LP6 so no changes to existing code should be required. In order to modify the number of significant digits that Lasso outputs the [Decimal->SetFormat] tag should be used.

Member Tags and Parentheses

Member tags which have multiple parameters must be surrounded by parentheses. Earlier versions of Lasso allowed some non-recommended syntax constructs to work. The new parser in Lasso Professional 7 is more strict about when parentheses are required around tag calls.

Specifically, the following syntax worked in Lasso Professional 6, but is no longer supported in Lasso Professional 7.

```
[Output: ((Date: '2003-12-01') -> Difference: (Date), -Hour)]
```

The code should be changed to the following. The parentheses around the [Date->Difference] tag clarify to which tag the -Hour parameter belongs.

```
[Output: ((Date: '2003-12-01') -> (Difference: (Date), -Hour))]
```

For best results any nested tags or member tags which require two or more parameters should be surrounded by parentheses.

PDF -Top Parameter

The -Top parameter in all PDF tags now always measures from the top margin of a document. In Lasso Professional 6 some of the PDF tags measured from the bottom of the page. See the LDML 7 Reference and the PDF chapter for additional details and a complete list of tags that have changed.

Global Variables

In Lasso Professional 7 global variables should always be manipulated using the [Global] tag rather than the [Variable] tag. The \$ symbol can be used to refer to either global variables or page variables. If both a page variable and a global variable are defined with the same name then the \$ symbol will return the value of the page variable.

Sites which do not use global variables do not require any modifications. The only sites that will require updates are those that used the [Variable] tag to refer to previously created global variables. These sites should be updated to use the [Global] tag instead.

[NSLookup]

Due to changes in Mac OS X 10.3 the [NSLookup] tag may not be able to perform reverse DNS lookups (from IP address to host name) on all DNS servers. Normal DNS lookups (from host name to IP address) should continue to work fine. This issue affects both Lasso Professional 6 and Lasso Professional 7 running on Mac OS X 10.3.

```
[NSLookup: '127.0.0.1']
```

[Repetition] Tag

The [Repetition] tag is deprecated in Lasso Professional 7 and will not be supported in the next version of Lasso. Converting loops that use the [Repetition] tag to use the modulus symbol % instead will result in faster code execution.

To update existing sites:

Replace the [Repetition: 2] tag with (Loop_Count % 2 == 0). The second operand of the % symbol should be whatever number was specified as a parameter to the [Repetition] tag.

For example, the following loop which makes use of [Repetition: 5] to display a message every fifth time through the loop.

```
[Loop: 100]
  [If: (Repetition: 5)]
    [Loop_Count] is divisible by 5!
  [/If]
[/Loop]
```

This loop can be rewritten using the modulus operator % as follows.

```
[Loop: 100]
  [If: (Loop_Count % 5 == 0)]
    [Loop_Count] is divisible by 5!
  [/If]
[/Loop]
```

The second loop will have exactly the same output as the first loop, but will run much faster.

[TCP_...] Tags

The [TCP_...] tags have been deprecated in favor of the new [Net] type and its member tags. Consult *Chapter 5: Advanced Programming Topics* in the Extending Lasso Guide for details about the new tags.

[Else:If] and [Else_If]

The [Else] tag supports the functionality that was provided by the dedicated [Else:If] and [Else_If] tags in prior versions of Lasso. In order to streamline the language and provide faster code processing only the [Else] tag is supported in Lasso Professional 7.

For example, in the following code the `[Else]` tag is used to check several condition. Without a conditional parameter the `[Else]` tag is the default value for the `[If] ... [/If]` tags and always returns its value.

```
[If: $Condition == 'Alpha']
... Alpha ...
[Else: $Condition == 'Beta']
... Beta ...
[Else: $Condition == 'Gamma']
... Gamma ...
[Else]
... Default ...
[/If]
```

To update existing sites:

Change all `[Else:If]` and `[Else_If]` tags to `[Else]`.

[LoopCount] and [LoopAbort]

In order to streamline the language and provide faster code processing the synonyms `[LoopCount]` for `[Loop_Count]` and `[LoopAbort]` for `[Loop_Abort]` are no longer supported in Lasso Professional 7.

To update existing sites:

Change all `[LoopCount]` tags to `[Loop_Count]` and all `[LoopAbort]` tags to `[Loop_Abort]`.

Container Tags

In order to provide more efficient code execution it is now necessary for all container tags to be defined in `LassoStartup`. Any container tags which are defined within included files or library files will no longer function properly.

The `[Define_Tag]` tag now accepts two parameters for creating container tags. If the `-Container` keyword is used then a simple, non-looping container tag will be created. If the `-Looping` keyword is used then a looping container tag will be created. The only difference is that the `[Loop_Count]` will only be modified in looping container tags.

See *Chapter 3: Custom Tags* in the *Extending Lasso Guide* for more details about defining custom container tags.

Custom Tags

All parameters and return values are now passed to custom tags by reference. Existing custom tags may need to be updated so that they do not cause any unwanted side effects or cause syntax errors.

The [PreCondition] and [PostCondition] tags are no longer supported. The -Type and -ReturnType parameters should be used in a custom tag definition in order to restrict the parameter types and return type for a custom tag.

Asynchronous custom tags do not have access to page variables from the page that called the custom tag. The documentation for LP6 was not clear on this point. Any variables which are required within the custom tag should be stored as globals or passed into the custom tag as parameters.

A number of other enhancements have been made to custom tags as well. See *Chapter 3: Custom Tags* in the Extending Lasso Guide for more details about defining custom tags.

To update existing sites for parameter passed by reference:

Use different names for locals defined within a custom tag and for the parameters of the tag. For example, the following tag will cause a syntax error since it is not possible to modify the incoming literal changing its type from an integer into a string.

```
[Define_Tag: 'Ex_UpperCase', -Required='Value']
  [Local: 'Value' = (String_UpperCase: 'Value')]
  [Return: #Value]
[/Define_Tag]

[Ex_UpperCase: 1] → Syntax Error
```

Instead, use a different name for the local variable within the tag. This code will work fine in Lasso Professional 7 and in Lasso Professional 6. By prefixing the local variables name with L_ there is no conflict with the incoming parameter names.

```
[Define_Tag: 'Ex_UpperCase', -Required='Value']
  [Local: 'L_Value' = (String_UpperCase: 'Value')]
  [Return: #L_Value]
[/Define_Tag]

[Ex_UpperCase: 1] → 1
```

To update existing sites to remove pre- and post-conditions:

Use the -Type and -ReturnType parameters to specify the types for each parameter of a custom tag and the return type for the tag. Additional error checking can be performed with the custom tag itself.

For example, the following custom tag definition uses [PreCondition] and [PostCondition] to check that all of the tag's parameters and the tag's return value are strings.

```
Define_Tag: 'Ex_Concatenate',
  -Required='Param1',
  -Required='Param2';
PreCondition: #Param1->Type == 'string';
PreCondition: #Param2->Type == 'string';
PostCondition: Return_Value == 'string';
Return: #Param1 + #Param2;
/Define_Tag;
```

In Lasso Professional 7 this tag can be rewritten as the following. The -Type parameters specify the required type for the preceding -Required parameter. The -ReturnType parameter specifies the required type for the return value. If the parameters or return type are not of the proper type then an error will be returned.

```
Define_Tag: 'Ex_Concatenate',
  -Required='Param1', -Type='string',
  -Required='Param2', -Type='string',
  -ReturnType='string';
Return: #Param1 + #Param2;
/Define_Tag;
```

XML Tags

The XML tags in Lasso Professional 7 have been re-implemented using native C/C++ libraries for greater speed and functionality. The behavior of some of the XML tags have changed and some modifications to existing sites may be required for full compatibility.

- [XML_Extract] – The [XML_Extract] tag will interpret some -XPath parameters differently. In particular, the new XML libraries interpret the XPath / to refer to the root of the XML data rather than the root tag in that data. /* can be used to refer to the root tag. The new [XML->Extract] tag is the preferred method of performing XPath on XML data and uses the same XPath syntax as [XML_Extract].
- [XML->Children] – The [XML->Children] tag now includes additional text children for many XML tags. These children represent the text on either side of embedded tags. For example, the following <a> tag has three children some, the tag, and text.

```
<a href="..."> Some <b>Embedded</b> Text </a>
```

LP6 would not provide access to these text children so the behavior of LP7 is preferred. The text children all have a name of text and may be empty if no text is specified between the various tags.

[Encode_ISOtoMac] and [Encode_MacToISO]

The [Encode_ISOtoMac] and [Encode_MacToISO] tags are not compatible with the Unicode strings that Lasso now uses to store strings. These tags must be modified in order for sites that use them to work properly with Lasso Professional 7.

To Update Existing Sites:

The output of the [Include_Raw], [File_Read], and other tags that might return data in a native character set have all been changed to the bytes type. The bytes type preserves the character set of the underlying data.

Note: See the earlier section on the *Bytes Type* for a full discussion of this new data type.

In LP6 a [File_Read] operation which read a Latin-1 (ISO 8859-1) file may have appeared like this. This code would translate the file from its native character set to Mac-Roman encoding.

```
[Variable: 'myFile' = (File_Read: 'myfile.text')]
[Variable: 'myString' = (Encode_ISOtoMac: $myFile)]
```

In LP7 the following code would be used. This code reads in the file as a byte stream and then uses [Bytes->ExportString] to convert the Latin-1 (ISO 8859-1) characters to the native Unicode-based double-byte strings that Lasso Professional 7 uses for character data.

```
[Variable: 'myFile' = (File_Read: 'myfile.text')]
[Variable: 'myString' = $myFile->(ExportString: 'iso8859-1')]
```

With this change the remainder of the code should not need to be changed since the end result has the same practical value, a natively encoded string.

Tag Name Changes (Lasso 5/6)

All tags from Lasso Professional 6 are supported in Lasso Professional 7 except for those listed in the table below. There are also a number of tag names which have changed or been deprecated in favor of new tags or methodologies in Lasso Professional 7.

The following table lists tags that are not supported in Lasso Professional 7. These tags must be replaced in order for sites to work properly in Lasso Professional 7.

Table 5: Unsupported Tags

LDML 6 Tag	LDML 7 Tag Equivalent
[Encode_MacToISO]	[Bytes->ExportString]
[Encode_ISOtoMac]	[Bytes->ExportString]

The following table lists the tag names that have been changed in Lasso Professional 7 since the release of Lasso Professional 5/6. The old versions of each tag will continue to work, but their use has been deprecated. Any new development in Lasso Professional 7 should use the new versions of the tag names.

Table 6: Tag Name Changes

LDML 6 Tag	LDML 7 Tag Equivalent
[Null->Up]	[Null->Parent]
[String->Length]	[String->Size]

The following table lists the tags from Lasso Professional 6 which have been deprecated in Lasso Professional 7 and what code equivalent should be used. The deprecated versions of these tags will continue to work, but any new development in Lasso Professional 7 should use the suggested code equivalent rather than the deprecated tags.

Table 7: Deprecated Tags

LDML 6 Tag	LDML 7 Tag Equivalent
[Date_GetCurrentDate]	[Date]
[Date_GetDay]	[Date->Day]
[Date_GetDayOfWeek]	[Date->DayOfWeek]
[Date_GetHour]	[Date->Hour]
[Date_GetMinute]	[Date->Minute]
[Date_GetMonth]	[Date->Month]
[Date_GetSecond]	[Date->Second]
[Date_GetYear]	[Date->Year]
[Error_NoRecordsFound]	Check for whether [Found_Count] is zero.
[PostCondition]	-ReturnType in [Define_Tag]
[PreCondition]	-Type or -Criteria in [Define_Tag]
[Repetition]	Modulus Symbol %
[TCP_Close]	[Net->Close]
[TCP_Open]	[Net->Connect]
[TCP_Send]	[Net->Read], [Net->Write]

Syntax Changes (Lasso 5)

Lasso Professional 7 introduces changes to some of the core syntax rules of LDML. Most of these changes were made to improve the reliability and error reporting of Lasso Professional 7. Some of these changes may require you to rewrite portions of your existing Lasso-based solutions for full compatibility with Lasso Professional 7. This section describes each change, why it was made and how to update existing format files.

Table 8: Syntax Changes

Syntax Change	Description
No Process Tags	New [NoProcess] ... [/NoProcess] tags allow a portion of a page to be passed to the browser without being processed.
Strict Syntax	A strict syntax option allows errors such as non-hyphenated parameters, non-quoted variables, and undefined tags to be reported as syntax errors.
Date Data Type	Date operations have been converted to a new date data type. LDML 5 date tags have some modifications.
Integer Rounding	The [Integer] tag now rounds to the nearest integer instead of truncating.
No Records Found	The [Error_NoRecordsFound] tag has been deprecated. Check whether [Found_Count] equals zero instead.

No Process Tags

Lasso Professional 7 includes a container tag [NoProcess] ... [/NoProcess] that instructs the LDML parser to ignore its contents. This allows code from other programming languages to be passed through to the browser without any processing by Lasso. These new tags do not require any changes to existing Lasso Web sites, but may make transitioning from older versions of Lasso easier.

The [NoProcess] ... [/NoProcess] tags must be embedded in a page exactly as written with no extra spaces or parameters within the square brackets. They cannot be used within LassoScript.

To instruct Lasso to ignore a portion of a page:

Use the [NoProcess] ... [/NoProcess] tags. In the following example, the entire contents of a JavaScript code block is ignored by Lasso. Any array references within the JavaScript will not be interpreted by Lasso as square bracketed tags.

```
[NoProcess]
  <script language="JavaScript">
    ... JavaScript Expressions ...
  </script>
[/NoProcess]
```

Strict Syntax

With strict syntax the following rules are enforced:

- All keyword parameters to built-in and custom tags must include a hyphen. This helps to find unknown tag parameters and to catch other common syntax errors.
- All string literals must be surrounded by quotes. This helps to prevent accidental calls to tags, to identify undefined variables, and to catch other common syntax errors.
- All tag calls must be defined. Unknown tags will no longer simply return the tag value as a string.

With strict syntax any of the errors above will be reported when a page is first loaded. They must be corrected before the code on the page will be executed. When upgrading to Lasso Professional 7 it is advisable to first try existing Lasso Professional 5 and 6 sites and correct any errors that are reported.

To update existing sites for strict syntax:

If a site is relatively small then the easiest method is to load each Web page and see if any errors are reported. The following tips can be used for a more methodical search.

- Check that all string literals are surrounded by quotes. Quotes are not necessary around integers or decimal numbers, hyphenated keyword parameters, tag names, or variable names when used with the & or # symbols.
- Check that all keywords in tag calls are preceded by a hyphen. Keyword and keyword/value parameters must be preceded by a hyphen, but do not need to be quoted. Name/value parameters should include quotes around both the name and value (unless they are numbers).
- Check that all command tags used within opening [Inline] tags are preceded by a hyphen. Quotes are not necessary around command tags, even when they are specified within an array.
- Check that all client-side JavaScript is formatted properly. JavaScript should either be included in [NoProcess] ... [/NoProcess] tags or HTML comment tags <!-- ... --> which ensure that no LDML code within is

processed. Or, any square brackets which are required within the JavaScript should be output from an [Output] tag.

[Output: 'array[4]]']

→ [array[4]]

Date Data Type

The date tags from LDML 5 have been replaced by new date and duration data types in LDML 7. This change should not require any changes to existing code, but many LDML 5 tags have been deprecated and many other operations are significantly easier using the new tags. See *Chapter 16: Date and Time Operations* for full documentation of the new date and duration data types.

Some highlights of the new date and duration data types include:

- The [Date] tag can be used in place of [Date_GetCurrent] date to return the current date and time.
- The [Date] tag now recognizes MySQL date formats natively as well as United States date formats.
- The [Date_Get...] tags have been replaced by member tags which perform equivalent functions. [Date->Day] returns the current day of the month and [Date->Year] returns the current 4-digit year.
- The week number can be output using [Date->Week] and the current day of the year can be output using [Date->DayOfYear].
- The duration between two dates can be output using the subtraction symbol [Output: (Date) - (Date: 3/4/1984)]. or a duration can be added to a date using the addition symbol [Output: (Date) + (Duration: -Hour=1)].
- The output format for all date tags on a format file can be set using [Date_SetFormat]. For example, [Date_SetFormat: '%Q %T'] will set all dates to output in MySQL date format.
- Individual dates can be formatted using [Date->Format]. For example [Date->(Format: '%Q %T')] will output the current date in MySQL date format.
- Upon casting a date type, Lasso 7 automatically adjusts invalid dates to be a valid equivalent, where Lasso 5 returns a null value instead of an invalid date. For example, 9/31/2002 is an invalid date because there are not 31 days in September. The expression [Date:'9/31/2002'] returns 10/1/2002 in Lasso 7, whereas [Date:'9/31/2002'] returns no value in Lasso 5.

Integer Rounding

The [Integer] tag now rounds decimal values to the nearest integer. In Lasso Professional 5 the [Integer] tag instead truncated decimal values to the next lowest integer. The new process yields a more accurate result. In general, no changes to existing sites should be necessary.

To update existing sites:

Use the [Math_Floor] tag to return the next lowest integer rather than using the [Integer] tag.

No Records Found

The [Error_NoRecordsFound] tag has been deprecated. This tag will continue to work with the Lasso Connector for FileMaker Pro, but may not work with MySQL databases or with third party data source connectors.

To update existing sites:

Change any code which uses [Error_NoRecordsFound] to instead check whether [Found_Count] is equal to zero. For example, the following code from LDML 5:

```
[If: (Error_CurrentError) == (Error_NoRecordsFound)]
    No records were found!
[/If]
```

Can be written as follows in LDML 7:

```
[If: (Found_Count) == 0]
    No records were found!
[/If]
```

Lasso MySQL (Lasso 5)

A number of changes have been made to the Lasso Connector for Lasso MySQL in order to make its behavior match that of the Lasso Connector for FileMaker Pro. These changes will not in general require any changes to existing Lasso Professional 5 sites.

Table 9: Lasso MySQL Syntax Changes

Syntax Change	Description
-Add and -Update	The -Add and -Update actions now return the record which was just added to the database or updated within the database by default.
Full Text Searching	The ft operator allows full text indices to be searched. Lasso Administration allows full text indices to be created.
Random Sorting	The -SortRandom keyword can be used to return MySQL results in random order.
Regular Expression Searching	The rx and nrx operators allow regular expression searches to be performed and all records which match or do not match the results to be returned.
Searching for Distinct Values	The -Distinct keyword allows only distinct records from search results to be returned.
Searching for Null Values	The inline tag now recognizes Null as a value distinct from the empty string allowing Null values in databases to be found.
Using LIMIT Options	The -UseLimit keyword instructs Lasso to use LIMIT options to select the found records to show rather than using native methods. This can result in better performance on large databases with large found sets.
Value Lists	Values lists are now supported for ENUM and SET data types within MySQL databases.

See *Chapter 9: MySQL Data Sources* for complete documentation of these changes.

Syntax Changes (Lasso WDE 3.x)

Lasso Professional 7 introduces changes to some of the core syntax rules of LDML. Some of these changes may require you to rewrite portions of your existing Lasso-based solutions. This section describes each change, why it was made and how to update existing format files.

Table 10: Syntax Changes

Syntax Change	Description
Square Brackets	All expressions in square brackets are now interpreted.
Commas	Commas are no longer allowed after tag names.
Keywords	Keyword names now always begin with a hyphen.

Encoding Keywords	The default is to HTML encode outermost substitution tags and apply no encoding to nested sub-tags.
Else If	The [Else:If] tag is no longer supported. The [Else] tag has been enhanced to provide the same functionality.
Include	The [Include] tag now returns an error if the specified file does not exist.
Post Inline	The [Post_Inline] tag has been replaced by a new scheduling facility accessed through the [Event_Schedule] tag.
SQL Inline	The [SQL_Inline] tag has been replaced by a new -SQL command tag which can be used in a normal [Inline] tag.
File Tags and Logging	The new distributed architecture means these tags work only on files accessible by Lasso Service.
Line Endings	The default line endings on Mac OS X are different from those for Mac OS 9.
JavaScript	Special care must be taken to ensure that array references in JavaScript are not interpreted by Lasso.
Macros	Macros are no longer supported. Much of their functionality can be achieved through custom tags.
Numeric Literals	Numeric literals must not be written with quotes. The conversion of strings to numeric values has changed.
Mathematical Precision	Precision is handled automatically by the new mathematical expressions and symbols and can be set explicitly using [Decimal->SetFormat] tag.
Double Quotes	Single quotes are preferred for designating string literals.
Restrictions	Restrictions on maximum values for math operations and looping tags have been eased.

Square Brackets

In earlier versions of Lasso, only tag names which were recognized by Lasso would be interpreted. In Lasso Professional 7, all square bracketed expressions are interpreted whether they contain a valid LDML tag or not. This allows expressions and member tags to be used within square brackets and allows custom tags to be used.

For example, the following expressions would all have been ignored in earlier versions of Lasso, but will be interpreted as indicated by Lasso Professional 7.

```
[45] → 45
[1 + 2] → 3
[blue] → blue
'aqua' + 'marine' → aquamarine
```

If square brackets are used decoratively on a page, e.g. to surround link names, they will be stripped out by Lasso Professional 7.

Note: See the section on JavaScript that follows for tips on using square brackets within client-side JavaScript contained in an LDML format file.

To update existing sites:

There are several options to update existing sites depending on how the square brackets are being used on a page.

- Use the HTML entities for square brackets. These include `[` for `[` and `]` for `]`. The following example would display a link name surrounded by square brackets.

```
<a href="default.lasso"> &#91; Home &#93; </a>
```

- Use the LDML `[Output]` tag to output an expression that includes square brackets. Lasso will not interpret the output of LDML tags so square brackets can be safely displayed on a page in this way. The following example would display a link name surrounded by square brackets.

```
<a href="default.lasso"> [Output: '[Home]'] </a>
```

The expression can also be written without the `[Output]` tag.

```
<a href="default.lasso"> ['[Home]'] </a>
```

Note: Any string literals which are output in this way should always be surrounded by single quotes, otherwise there is a danger that they might be interpreted as a tag.

- Create a custom tag that outputs text surrounded by square brackets. The following simple `[Define_Tag]` can be placed at the top of any page that requires it.

```
[Define_Tag: 'Bracket']
  [Output: (Params->(Get: 1))]
[/Define_Tag]
```

This tag can then be called to display a link name surrounded by square brackets.

```
<a href="default.lasso"> [Bracket: 'Home'] </a>
```

Commas

In earlier versions of Lasso, commas could optionally be used following the tag name, before any parameters of the tag. Although this syntax hasn't been recommended for some time there are still examples of it in the Lasso Web Data Engine 3.x documentation and in some Lasso-based Web sites.

The following example shows the tag construct with a comma following the tag name.

[Tag_Name, Parameters] (No longer supported)

This syntax was particularly common with tags that took only a single keyword. For example, both of the following tags were commonly written with a comma following the tag name.

[Server_Date, Short] (No longer supported)

[Error_CurrentError, ErrorCode] (No longer supported)

Using a colon after the tag name is now mandatory in Lasso Professional 7. This change was made in order to facilitate parsing of more complex expressions. The tag examples above must now be written as follows with a colon after the tag name. The following example also demonstrates the new method of specifying keyword names with a leading hyphen.

[Server_Date: -Short]
[Error_CurrentError: -ErrorCode]

To update existing sites:

Use a regular expression to correct format files that contain the older comma syntax. Most text editors and Web authoring environments can perform a find/replace using regular expressions.

- 1 Search for the following regular expression pattern to find tags in square brackets which have a comma after the tag name:

```
\([([A-Za-z_]+),([^\]]*)\)
```

Use this pattern as the replacement value:

```
[1:2]
```

- 2 Search for the following regular expression pattern to find sub-tags in parentheses which have a comma after the tag name:

```
\([([A-Za-z_]+),([^\)]*)\)
```

Use this pattern as the replacement value:

```
(1:2)
```

What the first regular expression does is search for a square bracket followed by a tag name, a comma, then any characters up until the closing square bracket. The replacement pattern inserts an opening square bracket, the tag name, a colon, the contents after the comma, and a final closing square bracket. The second regular expression performs the same steps with parentheses instead of square brackets.

Keywords

All keywords and keyword/value parameters (formerly named parameters) start with a hyphen in LDML 7. This used to be an option for command tags used within the [Inline] tag in Lasso Web Data Engine 3.x, but is now required for all tags. Most tags which were supported in Lasso Web Data Engine 3.x will continue to accept keywords without the leading hyphen so Lasso Web Data Engine 3.x solutions do not need to be rewritten. However, all keyword names without leading hyphens have been deprecated and are not guaranteed to work in future versions of Lasso.

This change was made so that LDML keywords can be clearly differentiated from user-defined name/value parameters and from tag names. This becomes especially important as users start to create custom tags which might have the same name as the keywords of existing tags.

To update existing sites:

- 1 Locate all keyword names that do not begin with a hyphen. For example, the following [Server_Date] tag contains both a tag-specific keyword and an encoding keyword, neither of which has been written with a hyphen:

```
[Server_Date: Short, EncodeNone]
```

The following [Inline] tag contains several command tags or keyword/value parameter that have not been written with hyphens:

```
[Inline:
  Database='Contacts',
  Table='People',
  'State'='WA',
  Search]
```

- 2 Change the keywords so their names start with a hyphen. The [Server_Date] tag is changed to the following with each keyword name beginning with a hyphen:

```
[Server_Date: -Short, -EncodeNone]
```

The [Inline] tag is changed to the following with each command tag and keyword/value parameter written with a hyphen:

```
[Inline:
  -Database='Contacts',
  -Table='People',
  'State'='WA',
  -Search]
```

- 3 Do not change user-defined name/value parameters. In the preceding example 'State'='WA' is not changed when updating the tag for compliance with Lasso Professional 7.

Note: The name ‘State’ has quotes around it in the preceding examples. All string literals should be specified with single quotes. This ensures that they will not be misidentified as a sub-tag or a keyword.

Encoding Keywords

The use of encoding keywords in substitution tags has been altered in Lasso Professional 7. All substitution tags which are used as sub-tags now have a default encoding of `-EncodeNone`. Only the outermost substitution tag (i.e. a tag in square brackets) has a default encoding of `-EncodeHTML`. This change was made in order to make Lasso easier to use for new users and to reduce the length of nested tag expressions.

The following example demonstrates the benefits of the new Lasso Professional 7 syntax. In LDML 3, the following `[String_Concatenate]` tag contains many sub-tag parameters which all have `EncodeNone` specified.

```
[String_Concatenate:
  (Field: 'First_Name', EncodeNone), '',
  (Field: 'Middle_Name', EncodeNone), '',
  (Field: 'Last_Name', EncodeNone)]
```

The preceding tag can be written as follows in LDML 7. Since the default encoding of each of the sub-tags is `-EncodeNone` the encoding keyword can be omitted. The resulting code is considerably shorter and easier to read.

```
[String_Concatenate: (Field: 'First Name'), '',
  (Field: 'Middle Name'), '', (Field: 'Last Name')]
```

The default encoding for the outermost tag in LDML 7 is still `-EncodeHTML` in order to maintain the security of sites powered by Lasso Professional 7. If a field is placed on a page without encoding then any JavaScript or HTML that the code contains will be live on the Web page. Only HTML from trusted sources should be allowed on your Web site.

LDML 7 includes additional encoding enhancements. Please see *Chapter 18: Encoding* for full details of how `[Encode_Set]` can be used to change the default encoding of a page and more.

Note: `-EncodeHTML` is now a valid encoding keyword which performs the same encoding as that which is performed if no encoding keyword is specified in an outermost substitution tag.

To update existing sites:

Encoding keywords still work as they did in Lasso Web Data Engine 3.x if they are specified in every tag. Existing code will generally work after an upgrade to Lasso Professional 7. However, the following use of encoding keywords will need to be rewritten.

- 1 Locate tags where the outermost tag has an `EncodeNone` encoding keyword and the sub-tags do not have any encoding keywords. For example, the following `[String_Concatenate]` tag has an `EncodeNone` keyword and the two `[Field]` tags do not have any encoding keywords.

```
[String_Concatenate: EncodeNone, (Field: 'First Name'), ' ', (Field: 'Last Name')]
```

- 2 Rewrite the tag by removing the `EncodeNone` keyword from the outermost tag. In the resulting LDML 7 code, no encoding keywords are required.

```
[String_Concatenate: (Field: 'First Name'), ' ', (Field: 'Last Name')]
```

Note: In the LDML 3 code, the `[Field]` sub-tags were automatically HTML encoded. The `EncodeNone` keyword in the outermost `[String_Concatenate]` tag ensured that double encoding was not applied. Since Lasso 7 does not encode sub-tags by default, the encoding keyword is no longer needed.

Else If

The `[Else:If:]` tag has been eliminated as a distinct tag, but the concept is still supported. `[Else:If: Condition]` is now syntactically equivalent to `[Else: (If: Condition)]` and the `[If]` and `[Else]` tags have been enhanced so that much of the old behavior of the `[Else:If:]` tag is preserved.

The following `[Else:If:]` tag will not work as expected in Lasso Professional 7 because the condition will be misinterpreted:

```
[Else:If: 'abc' == 'abc']
```

The condition will be interpreted as if the following tag had been written:

```
[Else: (If: 'abc') == 'abc']
```

The `(If: 'abc')` expression will return `True` and this will be compared to `'abc'`. Since `True` is not equal to `'abc'` this clause in the conditional will not be executed.

Note: If called individually, the `[If]` and `[Else]` tags will return the value of the specified conditional expressions parameter rather than returning an error about an unclosed container tag.

To update existing sites:

- Use parentheses around all conditional expressions. The following `[Else:If]` tag will work correctly in either Lasso Professional 7 or Lasso Web Data Engine 3.x:

```
[Else:If: ('abc' == 'abc')]
```

- Change the `[Else:If]` tag to `[Else]`. Lasso 7's `[Else]` tag has been enhanced so that it now works like the old `[Else:If]` tag if a condition is specified, but is still the marker for the default clause of the conditional if no condition

is specified. The following tag will work in Lasso Professional 7, but not in Lasso Web Data Engine 3.x:

```
[Else: 'abc' == 'abc']
```

Include

The `[Include]` tag now validates whether the specified file exists and returns an error if an invalid file path is specified. This means that programmatically constructed `[Include]` statements need to take a precaution so errors won't be shown to the site visitor.

To update existing sites:

The `[Protect]` ... `[/Protect]` tags can be used to suppress the error that is reported by the `[Include]` tag. The following code will not return an error, even though the file `fake.lasso` does not exist.

```
[Protect]
  [Include: 'fake.lasso']
[/Protect]
```

Post Inline

The `[Post_Inline]` tag is no longer supported in Lasso Professional 7. This tag relied on access to files which Lasso Service might not be able to locate because they could be on a separate machine. The replacement for `[Post_Inline]` is called `[Event_Schedule]` and has the following format:

```
[Event_Schedule:
  -Start=(Date, Defaults to Today),
  -End=(Date, Defaults to Never),
  -URL=(URL to Execute, Required)
  -Repeat=(True/False, Defaults to True if -Delay is set and False otherwise),
  -Restart=(True/False, Defaults to True),
  -Delay=(Minutes, Required if -Repeat is True),
  -Username=(Username for Authentication, Optional),
  -Password=(Password for Authentication, Optional)]
```

This tag schedules the execution of the response URL at a specific start date and time. The URL is fetched just as if a client had visited it through a Web browser. After the task is performed, it is optionally repeated a specified number of minutes later until the end date and time is reached. If the restart parameter is set to `True` then the repeating task will be rescheduled even after server restarts. Please see *Chapter 22: Control Tags* for complete documentation of the syntax of `[Event_Schedule]`.

To update existing sites:

Sites that rely on [Post_Inline] tags will need to be rewritten. The following steps must be taken:

- 1 Determine the URL of the post-inline response page you were calling.
- 2 Change the initial [Post_Inline] tag to the equivalent [Event_Schedule] tag using date calculations if necessary to determine the start date and time.
- 3 If the [Post_Inline] tag rescheduled itself in the response page then either the rescheduling call must be changed to an equivalent [Event_Schedule] tag or the automatic repeat feature of [Event_Schedule] can be used in its place.

SQL Inline

The [SQL_Inline] tag is no longer supported in Lasso Professional 7. This tag has been replaced by a more versatile -SQL command tag that can be used as the database action within any [Inline] tag.

```
[Inline: -SQL='...SQL Statement...']
... Inline Results ...
[/Inline]
```

The -SQL command tag can be used to issue SQL statements to the included Lasso MySQL data source or to any MySQL data source accessed through the Lasso Connector for MySQL. The -SQL command tag may also be supported by third party data source connectors. Please see *Chapter 9: MySQL Data Sources* for more information about using this tag.

To update existing sites:

Sites that rely on [SQL_Inline] tags will need to be rewritten. The following steps must be taken:

- 1 Change the opening and closing [SQL_Inline] ... [/SQL_Inline] tags to [Inline] ... [/Inline] tags. For example, following is a [SQL_Inline] that searches the People table of the Contacts database.

```
[SQL_Inline: Datasource='Contacts',
SQLStatement='SELECT First_Name, Last_Name from People']
...
[/SQL_Inline]
```

The first step is to change this to the following [Inline] ... [/Inline] tags, then to perform the remainder of the steps to complete the transformation.

```
[Inline: Datasource='Contacts',
SQLStatement='SELECT First_Name, Last_Name from People']
...
[/Inline]
```

- 2 Change the Datasource parameter to a -Database keyword/value parameter. Ensure that the database name is valid in the current Lasso Professional 7 setup.

```
[Inline: -Database='Contacts',
  SQLStatement='SELECT First_Name, Last_Name from People']
...
[/Inline]
```

Note: The ODBC data source module is not provided with Lasso Professional 7. Data sources must be available through the included Lasso Connector for MySQL or a third-party data source connector.

- 3 Change the SQLStatement parameter to a -SQL command tag. Change any table references within the SQL statement so they reference both the database and table name, not just the table name.

```
[Inline: -Database='Contacts',
  -SQL='SELECT First_Name, Last_Name from Contacts.People']
...
[/Inline]
```

- 4 If LDML tags are used within the SQL statement then they will need to be changed to expressions. In the following example, the name of the table is stored in a variable named MyTable and referenced using a square bracketed expression within the SQLStatement. This is no longer valid syntax.

```
[Var_Set: 'MyTable'='People']
[SQL_Inline: Datasource='Contacts',
  SQLStatement='SELECT First_Name, Last_Name from [Var: 'MyTable']']
...
[/SQL_Inline]
```

In Lasso Professional 7, this is changed to the following string expression that concatenates the value of the variable to the SQL statement explicitly.

```
[Variable: 'MyTable'='Contacts.Table']
[Inline: -Database='Contacts',
  -SQL='SELECT First_Name, Last_Name from ' + (Variable: 'MyTable')]
...
[/Inline]
```

Please see *Chapter 9: MySQL Data Sources* for more examples of creating SQL statements for use with the -SQL command tag and for information about how to display the results within the [Inline] ... [/Inline] tags.

File Tags and Logging

Lasso Professional 7 features a distributed architecture where Lasso Service can run on a different machine from the Web server on which Lasso Connector for IIS or Lasso Connector for Apache is installed. The file tags and logging tags can only manipulate files on the machine which is hosting Lasso Service. They have no access to the machine which is hosting a Lasso Web server connector.

If you are running both Lasso Service and your Web serving software on a single machine then no changes to existing file and logging tags should be necessary when you upgrade to Lasso Professional 7. Otherwise, please consult *Chapter 20: Files and Logging* for more information about how to access files in a two machine system.

Note: In contrast to the file and logging tags, the [Include] tag works exclusively with files from the Web serving machine. No changes should be necessary to your sites which use the [Include] tag unless you are using it to access log files or files which have been manipulated by the file tags. Use the [File_Read] tag for these situations.

Line Endings

Files created in Mac OS X, Windows 2000, or versions of the Mac OS 9 and earlier each have a different standard for line endings. This can cause confusion when moving files from one platform to another or from an earlier version of the Mac OS to Mac OS X. *Table 11: Line Endings* summarizes the different standards.

Table 11: Line Endings

Tag	Description
Mac OS X	Line feed: \n. Each line is ended with a single line feed character.
Mac OS 9 and Earlier	Carriage return: \r. Each line is ended with a single carriage return character.
Windows 2000	Line feed and carriage return: \r\n. Each line is ended with both a line feed and a carriage return character.

Line ending differences are handled automatically by Web servers and Web browsers so are generally only a concern when reading and writing files using the [File_...] tags. The following tips make working with files from different platforms easier.

- The default line endings used by the [File_LineCount] and [File_ReadLine] tags match the platform default. They are \n in Mac OS X

and `\r\n` in Windows 2000. The default for Lasso Web Date Engine 3.x's file tags on Mac OS 9 and earlier was `\r`.

- Specify line endings explicitly in the `[File_LineCount]` and `[File_ReadLine]` tags. For example, the following tag could be used to get the line count for a file that was originally created on Mac OS 9.

```
[File_LineCount: 'FileName.txt', -FileEndOfLine="\r"]
```

Or, the following tag could be used to get the line count for a file that was originally created on Windows 2000.

```
[File_LineCount: 'FileName.txt', -FileEndOfLine="\r\n"]
```

- Many FTP clients and Web browsers will automatically translate line endings when uploading or downloading files. Always check the characters which are actually used to end lines in a file. Don't assume that they will automatically be set to the standard of either the current platform or the platform from which they originated.
- A text editor such as Bare Bones BBEdit can be used to change the line endings in a file from one standard to another explicitly.

JavaScript

Since Lasso will interpret any expressions contained within square brackets special care must be taken to ensure that square brackets which are used for array accesses within client-side JavaScripts are not interpreted.

- Use the `[NoProcess]` ... `[/NoProcess]` tags to instruct Lasso not to interpret any of the code contained therein.

```
[NoProcess]
  <script language="JavaScript">
    ... JavaScript Expressions ...
  </script>
[/NoProcess]
```

- Lasso will not interpret any expressions that are contained within HTML comments. The following common method of surrounding a JavaScript with HTML comments ensures that neither Lasso nor older Web browsers will interpret the contents of the JavaScript.

```
<script language="JavaScript">
  <!--
    ... JavaScript Expressions ...
  // -->
</script>
```

The opening `<!--` expression is ignored by the JavaScript interpreter. The closing `-->` expression is formatted as part of a JavaScript comment by including it on a line starting with the JavaScript comment characters `//`.

- If LDML tags need to be used within a client-side JavaScript then the HTML comment can be opened and closed in order to allow Lasso to process portions of the JavaScript, but not others.

```
<script language="JavaScript">
  <!--
    ... JavaScript Expressions ...
  // -->
  var VariableName='[Output: ... LDML Expression ...]';
  <!--
    ... JavaScript Expressions ...
  // -->
</script>
```

- The LDML [Output] tag can be used to output short JavaScript segments that need to make use of square brackets. This technique is useful for JavaScript that is contained within the attributes of HTML tags or for JavaScripts that contain only a few square brackets.

In the following example, a select statement contains an [Output] tag in its onChange handler that returns a JavaScript expression containing square brackets to report which option was selected.

```
<select name="Select" multiple size="4"
  onChange="[Output: 'alert(this.options[this.selectedIndex])']">
  <option value="Value"> Value </option>
  ...
</select>
```

Macros

Macros are not supported in Lasso Professional 7. See the Extending Lasso Guide for information about rewriting macros as custom tags using the new [Define_Tag] tag in LDML 7.

Numeric Literals

In LDML 7 there is a distinction between number values and string values. This distinction makes advanced data type specific member tags and expression symbols possible. Strings are always enclosed in single quotes. Numbers are never enclosed in quotes. If you use quotes around a numeric literal then symbols which are used to manipulate that literal may assume it is a string.

For example, the following code specifies a mathematical operation, the numerical addition of 1 and 2:

```
[Output: 1 + 2] → 3
```

In contrast, the following code specifies a string operation, the string concatenation of the string '1' and the string '2', because the numbers are contained in quotes:

[Output: '1' + '2'] → 12

The legacy math and string tags from Lasso Web Data Engine 3.x still perform automatic type conversions on their arguments. This ensures that existing sites will not need to be rewritten. Both of the following tags return the same result despite the fact that the parameters are specified without quotes in one and with quotes in the other:

[Math_Add: 1, 2] → 3

[Math_Add: '1', '2'] → 3

When a string is converted into an integer or a decimal, only a number at the beginning of the string will be converted. For example, in the following conversion only the number 800 from the phone number will be output.

[Integer: '800-555-1212'] → 800

In earlier versions of Lasso all the numbers would have been extracted from the string yielding 8005551212 as the value. Existing sites may require modifications if this behavior was being counted on.

Note: Negative literals must be surrounded by parentheses when used on the right-hand side of two-operator symbols. For example, (1 + (-2)) or (\$Variable == (-4)).

Mathematical Precision

Mathematical symbols in LDML 7 do not have the same rounding behavior as math tags in LDML 3. For example, the following [Math_Div] tag returns a result with the LDML 7 standard of six significant digits instead of the maximum precision of its two parameters which it would have had in LDML 3.

[Math_Div: 10, 3.000] → 3.333333

In LDML 7 the mathematical symbols perform an integer operation if both parameters of the expression are integers. For example, the following division is performed and an integer result is returned:

[Output: 10 / 3] → 3

In LDML 7 the mathematical symbols perform a decimal operation if either of the parameters of the expression are a decimal value. Decimal results are always returned with at least six significant digits. For example, the following expressions return six significant digits of the result since one of the parameters is specified with a decimal point:

```
[Output: 10.0 / 3] → 3.333333
[Output: 10 / 3.0] → 3.333333
```

Existing sites should be modified to use the `[Math_Round]` tag or the `[Decimal->SetFormat]` tag to format results from mathematical expressions if less than six significant digits is desired.

The following example shows how to use `[Math_Round]` to reduce a division expression to three significant digits:

```
[Math_Round: (10.0 / 3.0), 1.000] → 3.333
```

The following example shows how to set a variable so it will always display three significant digits using the `[Decimal->SetFormat]` tag.

```
[Variable: 'Result' = (10.0 / 3.0)]
[(Variable: 'Result')->(SetFormat: -Precision=3)]
[Variable: 'Result']
```

→ 3.333

See *Chapter 15: Math Operations* for more information.

Double Quotes

Single quotes are preferred when specifying string literals. Double quotes are still supported, but have been deprecated. Double quotes are not guaranteed to work in the future. No changes to existing sites should be required, but all future development should use single quotes exclusively.

Restrictions

Some restrictions have been removed in LDML 7. Your site may need to be rewritten if it relied on one of these pre-defined restrictions. The following restrictions have been removed in LDML 7:

- Integer math now uses 64-bit values for greater precision. LDML 7 should support integer values up to 18,446,744,073,709,551,616. Decimal math and date calculation are also performed using 64-bit values.
- The `[Loop]` tag limit of 1000 iterations has been removed. It is now possible for infinite loops to occur in LDML so you may want to place your own upper limit on loop iterations as in the following code:

```
[Loop: 1000000]
  [If: (Loop_Count) > 1000][Loop_Abort][!If]
  ... Loop Contents ...
[/Loop]
```

Tag Name Changes (Lasso WDE 3.x)

In order to promote consistency in LDML 7 many tag names from LDML 3 had to be changed. The following chart details the tag names which have changed. Please consult the appropriate chapters in this book for more information about each individual tag name.

For the most part, these tag name changes will not require modifications to existing Lasso Web Data Engine 3.x sites. The old tag name is still supported in LDML 7. However, support for these old tag names is deprecated. They are not guaranteed to be supported in a future version of Lasso. All new development should take place using the new tag names.

Table 12: Command Tag Name Changes details the command tags which have changed in LDML 7. *Table 9: Substitution, Process, and Container Tag Name Changes* details the substitution, process, and container tags which have changed in LDML 7.

Table 12: Command Tag Name Changes

LDML 3 Tag	LDML 7 Tag Equivalent
-AddError	-ResponseAddError
-AddResponse	-ResponseAdd
-AnyError	-ResponseAnyError
-AnyResponse	-ResponseAny
-ClientPassword	-Password
-ClientUsername	-Username
-DeleteResponse	-ResponseDelete
-DoScript	-FMScript
-DoScript.Post	-FMScriptPost
-DoScript.Pre	-FMScriptPre
-DoScript.PreSort	-FMScriptPreSort
-DuplicateResponse	-ResponseDuplicate
-LogicalOperator	-OperatorLogical
-NoResultsError	-ResponseNoResultsError
-RequiredFieldMissingError	-ResponseRequiredFieldMissingError
-SecurityError	-ResponseSecurityError
-UpdateError	-ResponseUpdateError
-UpdateResponse	-ResponseUpdate

Table 13: Substitution, Process, and Container Tag Name Changes

LDML 3 Tag	LDML 7 Tag Equivalent
[Choice_List]	[Value_List]
[ChoiceListItem]	[Value_ListItem]
[DB_NameItem]	[Database_NameItem]
[DB_Names]	[Database_Names]
[DB_LayoutNameItem]	[Database_TableNameItem]
[DB_LayoutNames]	[Database_TableNames]
[Encode_Breaks]	[Encode_Break]
[File_LineCount]	[File_GetLineCount]
[Lasso_Abort]	[Abort]
[Lasso_Comment]	[Output_None]
[Lasso_Process]	[Process]
[Lasso_SessionID]	[Lasso_UniqueID]
[Link_Detail]	[Link_DetailURL]
[Logical_OperatorValue]	[Operator_LogicalValue]
[LoopAbort]	[Loop_Abort]
[LoopCount]	[Loop_Count]
[RandomNumber]	[Math_Random]
[RepeatingValueItem]	[Repeating_ValueItem]
[Roman]	[Math_Roman]
[SearchFieldItem]	[Search_FieldItem]
[SearchOptItem]	[Search_OptItem]
[SearchValueItem]	[Search_ValueItem]
[Shown_NextGroup]	[Link_NextGroup]
[Shown_NextGroupURL]	[Link_NextGroupURL]
[Shown_PrevGroup]	[Link_PrevGroup]
[Shown_PrevGroupURL]	[Link_PrevGroupURL]
[SortFieldItem]	[Sort_FieldItem]
[SortOrderItem]	[Sort_OrderItem]
[String_ToDecimal]	[Decimal]
[String_ToInteger]	[Integer]
[ValueListItem]	[Value_ListItem]

Unsupported Tags (Lasso WDE 3.x)

The following tags are no longer supported in LDML 7. If any of these tags are used in a Web site that was built for Lasso Web Data Engine 3.x they will need to be replaced before that Web site can be served by Lasso Professional 7. **Table 14: *Unsupported Tags*** is a complete list of tags that are not supported in LDML 7 including notes on how to update a Web site that relies on those tags for compatibility with Lasso Professional 7.

Table 14: Unsupported Tags

LDML 3 Tag	Notes
[4D_RefreshCache]	The 4D data source module is no longer provided.
[Apple_Event], [AE_...]	The Apple Event tags are no longer supported.
-DoScript.....Back	The -DoScript tags with a Back argument are no longer supported. Use the appropriate -FMScript... tag instead.
[Lasso_Datasources4D]	The 4D data source module is no longer provided.
[Lasso_DatasourcesODBC]	The ODBC data source module is no longer provided.
[Macro_...], -Macro	All macro tags are no longer supported. See the Extending Lasso Guide for information about custom tags.
[Post_Inline]	See the Post Inline section in this chapter for more information about how to convert [Post_Inline] calls to the [Event_Schedule] tag.
[Relation]	The [Relation] tag was equivalent to an [Inline] that performed a search in the related table.
-Scripts	This command tag only worked with the Apple Event based FileMaker Pro data source module. Use any command tag which performs a database action instead (e.g. -FindAll).
-Timeout	This command tag only worked with the Apple Event based FileMaker Pro data source module.
[Win_Exec]	This tag is no longer supported.

CDML Compatibility

Lasso Web Data Engine 3.x supported a number of CDML tags for compatibility with Web sites that were created for FileMaker Pro's Web Companion. These tags are no longer supported in Lasso Professional 7.

Early Lasso Compatibility

Lasso Web Data Engine 3.x supported a number of tags from earlier versions of Lasso for compatibility with sites that were created using the

earlier versions of Lasso. These tags are no longer supported in Lasso Professional 7.

FileMaker Pro (Lasso WDE 3.x)

Lasso Professional 7 includes Lasso Connector for FileMaker Pro which is the equivalent of the Lasso Web Data Engine 3.x FileMaker Pro Remote data source module. The functionality of the Apple Event based FileMaker Pro data source module is no longer supported since it was Mac specific and reliant upon the use of Apple Events.

If you were using the FileMaker Pro Remote data source module then no changes to your site should be required when you move the site over to Lasso Professional 7.

If you were not previously using the FileMaker Pro Remote data source module, some changes may be necessary. Lasso Connector for FileMaker Pro does not support the following features of the Apple Event based FileMaker Pro data source module from Lasso Web Data Engine 3.x.

- **Field-Level Search Operators** are not supported. The `-OperatorBegin` and `-OperatorEnd` tags cannot be used to create complex queries with a FileMaker Pro database.
- **Automatic Image Conversion** is not supported for PICT images stored in FileMaker Pro container field. However, GIFs and JPEGs stored in container field can be retrieved. The parameters of the `[Image_URL]` tag are ignored and images are served in the format stored in the database.
- Certain **Script Command Tags** are not supported including `-DoScript.Back`, `-DoScript.Post.Back`, `-DoScript.PreSort.Back`, `-DoScript.Pre.Back`. These tags all instruct FileMaker Pro to send itself to the background after the script is completed. Use the `-FMScript` commands without the `Back` argument instead.
- **FileMaker Pro 3** is no longer supported since this version does not provide the Web Companion necessary to make a remote connection to FileMaker Pro.

However, in exchange for the omissions there are some advantages to using Lasso Connector for FileMaker Pro.

- FileMaker Pro can be accessed via TCP/IP on the same machine or on a different machine.
- Multiple FileMaker Pro applications running on different machines can be accessed from a single installation of Lasso Professional 7.
- The `-ReturnField` tag allows you to limit the fields that are returned from a search or other database action.

- GIFs and JPEGs can be stored in FileMaker Pro container fields and served directly without any conversion.

Upgrading FileMaker Pro Based Sites

If a site was created using the FileMaker Remote data source module then no changes should be necessary when moving the site to Lasso Professional 7. Simply follow the instructions in *Chapter 13: Upgrading* in the Lasso Professional 7 Setup Guide in order to configure Lasso Connector for FileMaker Pro to point to the appropriate FileMaker Pro Web Companion.

If a site was created using the Apple Event based FileMaker Pro data source module or relied on FileMaker Pro 3.x then the following changes will need to be made in order to ensure that the site is compatible with Lasso Professional 7.

To upgrade a FileMaker Pro based site:

- 1 A site that relies on FileMaker Pro 3 will need to be upgraded to FileMaker Pro 4.x or FileMaker Pro Unlimited 5.x.
- 2 Configure FileMaker Pro Web Companion according to the instructions in *Chapter 5: Data Sources* of the Lasso Professional 7 Setup Guide. The Web Companion needs to be activated and all databases that are to be shared need to have their Sharing... settings established.
- 3 Modify any database searches that relied on the -OperatorBegin and -OperatorEnd command tags so that they no longer reference these tags.
- 4 Modify any calls to -DoScript... to call one of the new -FMScript... equivalents. Any database action that relies on the -Scripts command tag needs to be rewritten with a database action such as -FindAll.
- 5 Ensure that the images stored in container fields are either GIFs or JPEGs. These images will be served directly by the Web Companion.

A

Appendix A

LDML 7 Tag List

This appendix contains two charts which summarize the language of Lasso Professional 7, e.g. Lasso Dynamic Markup Language (LDML 7).

- **LDML 7 Tag List** contains all of the preferred tags in LDML 7 and lists the tag type (Command, Container, Member, Process, or Substitution), status (Preferred, Synonym, or Abbreviation), change since LDML 3 (None, None, or None), and the page number in the Lasso 7 Language Guide where the tag is principally documented.
- **LDML 7 Legacy Tag List** contains all of the deprecated tags which are still supported in LDML 7, but may not be supported in a future version of Lasso.

Table Key: Tags which have the * symbol by the page number are a synonym or abbreviation of a tag which is documented on the specified page. Tags which have the ‡ symbol by the page number are documented on the specified page in the Extending Lasso Guide.

Note: The tag listings are also available in the LDML 7 Reference which provides different list views, allows tags to be searched, and includes additional details about each tag in LDML 7.

Upgrading Note: See *Chapter 32: Upgrading Your Solutions* for a list of LDML 3 tags that are not supported in LDML 7 and for specific tips regarding how to upgrade solutions that rely on deprecated tags.

LDML 7 Tag List

Action

Tag	Type	Status	Change	Page
-Add	Command	Preferred	None	110
-Delete	Command	Preferred	None	110
-Duplicate	Command	Preferred	None	110
-FindAll	Command	Preferred	None	110
-Image	Command	Preferred	None	110
-Nothing	Command	Preferred	None	110
-Random	Command	Preferred	None	110
-Search	Command	Preferred	None	110
-Show	Command	Preferred	None	110
-SQL	Command	Preferred	None	110
-Update	Command	Preferred	None	110

Administration

Tag	Type	Status	Change	Page
[Admin_ChangeUser]	Substitution	Preferred	None	428
[Admin_CreateUser]	Substitution	Preferred	None	428
[Admin_CurrentGroups]	Substitution	Preferred	New	428
[Admin_CurrentUsername]	Substitution	Preferred	New	428
[Admin_GroupAssignUser]	Substitution	Preferred	None	428
[Admin_GroupListUsers]	Substitution	Preferred	None	428
[Admin_GroupRemoveUser]	Substitution	Preferred	None	428
[Admin_LassoServicePath]	Substitution	Preferred	New	428
[Admin_ListGroups]	Substitution	Preferred	None	428
[Admin_RefreshSecurity]	Substitution	Preferred	New	428
[Admin_ReloadDataSource]	Substitution	Preferred	New	428
[Auth]	Process	Preferred	None	426
[Auth_Admin]	Process	Preferred	None	426
[Auth_Custom]	Process	Preferred	New	426
[Auth_Group]	Process	Preferred	New	426
[Auth_User]	Process	Preferred	New	426
[Lasso_DatasourcesFileMaker]	Substitution	Preferred	None	215
[Lasso_DatasourcesLassoMySQL]	Substitution	Preferred	None	189
[Lasso_DatasourcesMySQL]	Substitution	Preferred	None	189
[Lasso_DatasourceModuleName]	Substitution	Preferred	None	441
[Lasso_TagExists]	Substitution	Preferred	None	441
[Lasso_TagModuleName]	Substitution	Preferred	None	441
[Lasso_Version]	Substitution	Preferred	None	441
[Log_SetDestination]	Process	Preferred	None	391
[Tags]	Substitution	Preferred	None	439

Array

Tag	Type	Status	Change	Page
[Array]	Substitution	Preferred	None	340
[Array->FindIndex]	Member	Preferred	None	342
[Array->Find]	Member	Preferred	None	342
[Array->Get]	Member	Preferred	None	342
[Array->Insert]	Member	Preferred	None	342
[Array->Join]	Member	Preferred	None	342
[Array->Last]	Member	Preferred	None	342
[Array->Merge]	Member	Preferred	None	342
[Array->Remove]	Member	Preferred	None	342
[Array->RemoveAll]	Member	Preferred	None	342
[Array->Size]	Member	Preferred	None	342
[Array->Sort]	Member	Preferred	None	342
[Map]	Substitution	Preferred	None	352
[Map->Find]	Member	Preferred	None	353
[Map->Get]	Member	Preferred	None	353
[Map->Insert]	Member	Preferred	None	353
[Map->Keys]	Member	Preferred	None	353
[Map->Remove]	Member	Preferred	None	353
[Map->Size]	Member	Preferred	None	353
[Map->Values]	Member	Preferred	None	353
[Pair]	Substitution	Preferred	None	357
[Pair->First]	Member	Preferred	None	357
[Pair->Get]	Member	Preferred	None	357*
[Pair->Name]	Member	Synonym	None	357*
[Pair->Second]	Member	Preferred	None	357
[Pair->Size]	Member	Preferred	None	357*
[Pair->Value]	Member	Synonym	None	357*

Client

Tag	Type	Status	Change	Page
[Client_Address]	Substitution	Preferred	None	508
[Client_Browser]	Substitution	Preferred	None	508
[Client_ContentLength]	Substitution	Preferred	None	507
[Client_ContentType]	Substitution	Preferred	None	507
[Client_CookieList]	Substitution	Preferred	None	492
[Client_Cookies]	Substitution	Preferred	None	492
[Client_FormMethod]	Substitution	Preferred	None	507
[Client_GETArgs]	Substitution	Preferred	None	507
[Client_GETParams]	Substitution	Preferred	None	507
[Client_Headers]	Substitution	Preferred	None	507
[Client_IP]	Substitution	Preferred	None	508
[Client_Password]	Substitution	Preferred	None	507
[Client_POSTArgs]	Substitution	Preferred	None	507
[Client_POSTParams]	Substitution	Preferred	None	507

[Client_Type]	Substitution	Preferred	None	508
[Client_Username]	Substitution	Preferred	None	507
[WAP_IsEnabled]	Substitution	Preferred	None	515
[WAP_MaxButtons]	Substitution	Preferred	None	515
[WAP_MaxColumns]	Substitution	Preferred	None	515
[WAP_MaxHorzPixels]	Substitution	Preferred	None	515
[WAP_MaxRows]	Substitution	Preferred	None	515
[WAP_MaxVertPixels]	Substitution	Preferred	None	515

Conditional

Tag	Type	Status	Change	Page
[Abort]	Process	Preferred	None	274
[Case]	Substitution	Preferred	None	267
[Else]	Substitution	Preferred	None	265
[If]	Container	Preferred	None	265
[Iterate]	Container	Preferred	None	273
[Loop]	Container	Preferred	None	270
[Loop_Abort]	Process	Preferred	None	270
[Loop_Count]	Substitution	Preferred	None	270
[Select]	Container	Preferred	None	267
[While]	Container	Preferred	None	274

Custom Tag

Tag	Type	Status	Change	Page
[Define_Tag]	Container	Preferred	None	33‡
[Define_Type]	Container	Preferred	None	62‡
[Params]	Substitution	Preferred	None	33‡
[Params_Up]	Substitution	Preferred	None	33‡
[Return]	Process	Preferred	None	33‡
[Run_Children]	Substitution	Preferred	None	33‡
[Self]	Substitution	Preferred	None	62‡
[Tag_Name]	Substitution	Preferred	None	33‡

Database

Tag	Type	Status	Change	Page
[Checked]	Substitution	Preferred	None	226
[Column]	Substitution	Synonym	None	125*
[Column_Name]	Substitution	Synonym	None	127*
-Database	Command	Preferred	None	123
[Database_ChangeColumn]	Process	Preferred	None	203*
[Database_ChangeField]	Process	Preferred	None	203
[Database_CreateColumn]	Process	Preferred	None	203*
[Database_CreateField]	Process	Preferred	None	203
[Database_CreateTable]	Process	Preferred	None	203
[Database_Name]	Substitution	Preferred	None	120

[Database_NameItem]	Substitution	Preferred	None	127
[Database_Names]	Container	Preferred	None	127
[Database_RealName]	Substitution	Preferred	None	131
[Database_RemoveColumn]	Process	Preferred	None	203*
[Database_RemoveField]	Process	Preferred	None	203
[Database_RemoveTable]	Process	Preferred	None	203
[Database_SchemaNameItem]	Substitution	Preferred	None	240
[Database_SchemaNames]	Container	Preferred	None	240
[Database_TableNameItem]	Substitution	Preferred	None	127
[Database_TableNames]	Container	Preferred	None	127
-Distinct	Command	Preferred	None	192
[Field]	Substitution	Preferred	None	125
[Field_Name]	Substitution	Preferred	None	127
[Field_Names]	Substitution	Preferred	New	127
-FMScript	Command	Preferred	None	234
-FMScriptPost	Command	Synonym	None	234*
-FMScriptPre	Command	Preferred	None	234
-FMScriptPreSort	Command	Preferred	None	234
-KeyColumn	Command	Synonym	None	123*
[KeyColumn_Name]	Substitution	Synonym	None	120*
[KeyColumn_Value]	Substitution	Synonym	None	120*
-KeyField	Command	Preferred	None	123
[KeyField_Name]	Substitution	Preferred	None	120
[KeyField_Value]	Substitution	Preferred	None	120
-KeyValue	Command	Preferred	None	123
-Layout	Command	Synonym	None	123*
[Layout_Name]	Substitution	Synonym	None	120*
-MaxRecords	Command	Preferred	None	123
-MaxRows	Command	Synonym	None	123*
[Option]	Substitution	Preferred	None	226
[Portal]	Container	Preferred	None	218
[Records]	Container	Preferred	None	125
[Records_Array]	Substitution	Preferred	New	125
[Repeating]	Container	Preferred	None	218
[Repeating_ValueItem]	Substitution	Preferred	None	218
-Req	Command	Abbreviation	None	123*
-Required	Command	Preferred	None	123
-ReturnColumn	Command	Synonym	None	123*
-ReturnField	Command	Preferred	None	123
[Rows]	Container	Synonym	None	125*
[Rows_Array]	Substitution	Preferred	New	125
-Schema	Command	Preferred	None	240
[Schema_Name]	Substitution	Preferred	None	240
[Selected]	Substitution	Preferred	None	226
-SkipRecords	Command	Preferred	None	123
-SkipRows	Command	Synonym	None	123*
-SortColumn	Command	Synonym	None	123*
-SortField	Command	Preferred	None	123

-SortOrder	Command	Preferred	None	123
-SortRandom	Command	Preferred	None	192
-Table	Command	Preferred	None	123
[Table_Name]	Substitution	Preferred	None	120
[Table_RealName]	Substitution	Preferred	None	131
-UseLimit	Command	Preferred	None	192
[Value_List]	Container	Preferred	None	226
[Value_ListItem]	Substitution	Preferred	None	226

Date

Tag	Type	Status	Change	Page
[Date]	Substitution	Preferred	Updated	325
[Date->Add]	Member	Preferred	None	335
[Date->Day]	Member	Preferred	None	330
[Date->DayofWeek]	Member	Preferred	None	330
[Date->DayofYear]	Member	Preferred	None	330
[Date->Difference]	Member	Preferred	None	335
[Date->Format]	Member	Preferred	None	329
[Date->GMT]	Member	Preferred	None	330
[Date->Hour]	Member	Preferred	None	330
[Date->Millisecond]	Member	Preferred	None	330
[Date->Minute]	Member	Preferred	None	330
[Date->Month]	Member	Preferred	None	330
[Date->Second]	Member	Preferred	None	330
[Date->SetFormat]	Member	Preferred	None	329
[Date->Subtract]	Member	Preferred	None	335
[Date->Time]	Member	Preferred	None	330
[Date->Week]	Member	Preferred	None	330
[Date->Year]	Member	Preferred	None	330
[Date_Add]	Substitution	Preferred	None	334
[Date_Difference]	Substitution	Preferred	None	334
[Date_Format]	Substitution	Preferred	None	325
[Date_GetLocalTimeZone]	Substitution	Preferred	None	325
[Date_GMTtoLocal]	Substitution	Preferred	None	325
[Date_LocaltoGMT]	Substitution	Preferred	None	325
[Date_Maximum]	Substitution	Preferred	None	325
[Date_Minimum]	Substitution	Preferred	None	325
[Date_SetFormat]	Process	Preferred	None	325
[Date_Subtract]	Substitution	Preferred	None	334
[Duration]	Substitution	Preferred	None	332
[Duration->Day]	Member	Preferred	None	332
[Duration->Hour]	Member	Preferred	None	332
[Duration->Minute]	Member	Preferred	None	332
[Duration->Month]	Member	Preferred	None	332

[Duration->Second]	Member	Preferred	None	332
[Duration->Week]	Member	Preferred	None	332
[Duration->Year]	Member	Preferred	None	332
[Server_Date]	Substitution	Preferred	None	445
[Server_Day]	Substitution	Preferred	None	445
[Server_Time]	Substitution	Preferred	None	445

Encoding

Tag	Type	Status	Change	Page
[Decode_Base64]	Substitution	Preferred	None	367
[Decode_HTML]	Substitution	Preferred	New	367
[Decode_URL]	Substitution	Preferred	None	367
[Encode_Base64]	Substitution	Preferred	None	367
[Encode_Break]	Substitution	Preferred	None	367
[Encode_HTML]	Substitution	Preferred	None	367
[Encode_Set]	Container	Preferred	None	366
[Encode_Smart]	Substitution	Preferred	None	367
[Encode_SQL]	Substitution	Preferred	None	367
[Encode_StrictURL]	Substitution	Preferred	None	367
[Encode_URL]	Substitution	Preferred	None	367
[Encode_XML]	Substitution	Preferred	None	367

Encryption

Tag	Type	Status	Change	Page
[Decrypt_BlowFish]	Substitution	Preferred	None	368
[Encrypt_BlowFish]	Substitution	Preferred	None	368
[Encrypt_MD5]	Substitution	Preferred	None	368

Error

Tag	Type	Status	Change	Page
[Error_AddError]	Substitution	Preferred	None	417
[Error_Code]	Substitution	Preferred	New	415
[Error_ColumnRestriction]	Substitution	Synonym	None	417*
[Error_CurrentError]	Substitution	Preferred	None	415
[Error_DatabaseConnectionUnavailable]	Substitution	Preferred	None	417
[Error_DatabaseTimeout]	Substitution	Preferred	None	417
[Error_DeleteError]	Substitution	Preferred	None	417
[Error_FieldRestriction]	Substitution	Preferred	None	417
[Error_FileNotFound]	Substitution	Preferred	None	417
[Error_InvalidDatabase]	Substitution	Preferred	None	417
[Error_InvalidPassword]	Substitution	Preferred	None	417
[Error_InvalidUsername]	Substitution	Preferred	None	417
[Error_ModuleNotFound]	Substitution	Preferred	None	417
[Error_Msg]	Substitution	Preferred	New	415

[Error_NoError]	Substitution	Preferred	None	417
[Error_NoPermission]	Substitution	Preferred	None	417
[Error_OutOfMemory]	Substitution	Preferred	None	417
[Error_ReqColumnMissing]	Substitution	Abbreviation	None	417*
[Error_ReqFieldMissing]	Substitution	Abbreviation	None	417
[Error_RequiredColumnMissing]	Substitution	Synonym	None	417*
[Error_RequiredFieldMissing]	Substitution	Preferred	None	417*
[Error_SetErrorCode]	Process	Preferred	None	415
[Error_SetErrorMessage]	Process	Preferred	None	415
[Error_UpdateError]	Substitution	Preferred	None	417
[Fail]	Process	Preferred	None	419
[Fail_If]	Process	Preferred	None	419
[Handle]	Container	Preferred	None	419
[Handle_Error]	Container	Preferred	New	419
[Protect]	Container	Preferred	None	419

File

Tag	Type	Status	Change	Page
[File]	Substitution	Preferred	New	404
[File->Close]	Member	Preferred	New	405
[File->Delete]	Member	Preferred	New	405
[File->GetPosition]	Member	Preferred	New	405
[File->Get]	Member	Preferred	New	405
[File->IsOpen]	Member	Preferred	New	405
[File->MoveTo]	Member	Preferred	New	405
[File->Name]	Member	Preferred	New	405
[File->Open]	Member	Preferred	New	405
[File->Path]	Member	Preferred	New	405
[File->Read]	Member	Preferred	New	405
[File->SetMode]	Member	Preferred	New	405
[File->SetPosition]	Member	Preferred	New	405
[File->SetSize]	Member	Preferred	New	405
[File->Size]	Member	Preferred	New	405
[File->Write]	Member	Preferred	New	405
[File_Copy]	Process	Preferred	None	396
[File_Create]	Process	Preferred	None	396
[File_CreationDate]	Substitution	Preferred	None	396
[File_CurrentError]	Substitution	Preferred	None	396
[File_Delete]	Process	Preferred	None	396
[File_Exists]	Substitution	Preferred	None	396
[File_GetLineCount]	Substitution	Preferred	None	396
[File_GetSize]	Substitution	Preferred	None	396
[File_IsDirectory]	Substitution	Preferred	None	396
[File_LineCount]	Substitution	Preferred	None	396
[File_ListDirectory]	Substitution	Preferred	None	396
[File_ModDate]	Substitution	Preferred	None	396
[File_Move]	Process	Preferred	None	396

[File_Read]	Substitution	Preferred	None	396
[File_ReadLine]	Substitution	Preferred	None	396
[File_Rename]	Process	Preferred	None	396
[File_SetSize]	Process	Preferred	None	396
[File_Uploads]	Substitution	Preferred	None	396
[File_Write]	Process	Preferred	None	396

Image

Tag	Type	Status	Change	Page
[Image]	Substitution	Preferred	New	464
[Image->AddComment]	Member	Preferred	New	467
[Image->Annotate]	Member	Preferred	New	473
[Image->Blur]	Member	Preferred	New	470
[Image->Comments]	Member	Preferred	New	465
[Image->Composite]	Member	Preferred	New	474
[Image->Contrast]	Member	Preferred	New	470
[Image->Crop]	Member	Preferred	New	469
[Image->Depth]	Member	Preferred	New	465
[Image->Describe]	Member	Preferred	New	465
[Image->Enhance]	Member	Preferred	New	470
[Image->Execute]	Member	Preferred	New	476
[Image->File]	Member	Preferred	New	465
[Image->FlipH]	Member	Preferred	New	469
[Image->FlipV]	Member	Preferred	New	469
[Image->Format]	Member	Preferred	New	465
[Image->Height]	Member	Preferred	New	465
[Image->Modulate]	Member	Preferred	New	465
[Image->Pixel]	Member	Preferred	New	470
[Image->ResolutionH]	Member	Preferred	New	465
[Image->ResolutionV]	Member	Preferred	New	465
[Image->Rotate]	Member	Preferred	New	465
[Image->Save]	Member	Preferred	New	469
[Image->Scale]	Member	Preferred	New	469
[Image->Sharpen]	Member	Preferred	New	469
[Image->Width]	Member	Preferred	New	470
[Image_URL]	Substitution	Preferred	None	231
[IMG]	Substitution	Preferred	None	231

Include

Tag	Type	Status	Change	Page
[Include]	Substitution	Preferred	None	386
[Include_Raw]	Substitution	Preferred	None	386
[Include_URL]	Substitution	Preferred	None	484
[Library]	Process	Preferred	None	386

Link

Tag	Type	Status	Change	Page
[Link]	Container	Preferred	New	157
[Link_CurrentAction]	Container	Preferred	None	161
[Link_CurrentActionParams]	Substitution	Preferred	New	162
[Link_CurrentActionURL]	Substitution	Preferred	None	160
[Link_CurrentGroup]	Container	Preferred	None	161
[Link_CurrentGroupParams]	Substitution	Preferred	New	162
[Link_CurrentGroupURL]	Substitution	Preferred	None	160
[Link_CurrentRecord]	Container	Preferred	None	161
[Link_CurrentRecordParams]	Substitution	Preferred	New	162
[Link_CurrentRecordURL]	Substitution	Preferred	None	160
[Link_Detail]	Container	Preferred	None	161
[Link_DetailParams]	Substitution	Preferred	New	162
[Link_DetailURL]	Substitution	Preferred	None	160
[Link_FirstGroup]	Container	Preferred	None	161
[Link_FirstGroupParams]	Substitution	Preferred	New	162
[Link_FirstGroupURL]	Substitution	Preferred	None	160
[Link_FirstRecord]	Container	Preferred	None	161
[Link_FirstRecordParams]	Substitution	Preferred	New	162
[Link_FirstRecordURL]	Substitution	Preferred	None	160
[Link_LastGroup]	Container	Preferred	None	161
[Link_LastGroupParams]	Substitution	Preferred	New	162
[Link_LastGroupURL]	Substitution	Preferred	None	160
[Link_LastRecord]	Container	Preferred	None	161
[Link_LastRecordParams]	Substitution	Preferred	New	162
[Link_LastRecordURL]	Substitution	Preferred	None	160
[Link_NextGroup]	Container	Preferred	None	161
[Link_NextGroupParams]	Substitution	Preferred	New	162
[Link_NextGroupURL]	Substitution	Preferred	None	160
[Link_NextRecord]	Container	Preferred	None	161
[Link_NextRecordParams]	Substitution	Preferred	New	162
[Link_NextRecordURL]	Substitution	Preferred	None	160
[Link_Params]	Substitution	Preferred	New	157
[Link_PrevGroup]	Container	Preferred	None	161
[Link_PrevGroupParams]	Substitution	Preferred	New	162
[Link_PrevGroupURL]	Substitution	Preferred	None	160
[Link_PrevRecord]	Container	Preferred	None	161
[Link_PrevRecordParams]	Substitution	Preferred	New	162
[Link_PrevRecordURL]	Substitution	Preferred	None	160
[Link_URL]	Substitution	Preferred	New	157
[Referer]	Container	Synonym	None	161
[Referer_URL]	Substitution	Synonym	None	160
[Referrer]	Container	Preferred	None	161
[Referrer_URL]	Substitution	Preferred	None	160

Literal

Tag	Type	Status	Change	Page
[False]	Substitution	Preferred	None	275*
[True]	Substitution	Preferred	None	275*

Math

Tag	Type	Status	Change	Page
[Currency]	Substitution	Preferred	New	320
[Decimal]	Substitution	Preferred	None	307
[Decimal->SetFormat]	Member	Preferred	None	311
[Integer]	Substitution	Preferred	None	306
[Integer->BitAnd]	Member	Preferred	None	313
[Integer->BitClear]	Member	Preferred	None	313
[Integer->BitFlip]	Member	Preferred	None	313
[Integer->BitNot]	Member	Preferred	None	313
[Integer->BitOr]	Member	Preferred	None	313
[Integer->BitSet]	Member	Preferred	None	313
[Integer->BitShiftLeft]	Member	Preferred	None	313
[Integer->BitShiftRight]	Member	Preferred	None	313
[Integer->BitTest]	Member	Preferred	None	313
[Integer->BitXOr]	Member	Preferred	None	313
[Integer->SetFormat]	Member	Preferred	None	313
[Locale_Format]	Substitution	Preferred	New	320
[Math_Abs]	Substitution	Preferred	None	316
[Math_ACos]	Substitution	Preferred	None	319
[Math_Add]	Substitution	Preferred	None	316
[Math_ASin]	Substitution	Preferred	None	319
[Math_ATan]	Substitution	Preferred	None	319
[Math_ATan2]	Substitution	Preferred	None	319
[Math_Ceil]	Substitution	Preferred	None	316
[Math_ConvertEuro]	Substitution	Preferred	None	316
[Math_Cos]	Substitution	Preferred	None	319
[Math_Div]	Substitution	Preferred	None	316
[Math_Exp]	Substitution	Preferred	None	319
[Math_Floor]	Substitution	Preferred	None	316
[Math_Ln]	Substitution	Preferred	None	319
[Math_Log]	Substitution	Preferred	None	319
[Math_Log10]	Substitution	Preferred	None	319
[Math_Max]	Substitution	Preferred	None	316
[Math_Min]	Substitution	Preferred	None	316
[Math_Mod]	Substitution	Preferred	None	316
[Math_Mult]	Substitution	Preferred	None	316
[Math_Pow]	Substitution	Preferred	None	319
[Math_Random]	Substitution	Preferred	None	316
[Math_RInt]	Substitution	Preferred	None	316
[Math_Roman]	Substitution	Preferred	None	316

[Math_Round]	Substitution	Preferred	None	316
[Math_Sin]	Substitution	Preferred	None	319
[Math_Sqrt]	Substitution	Preferred	None	319
[Math_Sub]	Substitution	Preferred	None	316
[Math_Tan]	Substitution	Preferred	None	319
[Percent]	Substitution	Preferred	New	320
[Scientific]	Substitution	Preferred	New	320

Networking

Tag	Type	Status	Change	Page
[Net]	Substitution	Preferred	New	108‡
[Net->Accept]	Member	Preferred	New	110‡
[Net->Bind]	Member	Preferred	New	109‡
[Net->Close]	Member	Preferred	New	109‡
[Net->Connect]	Member	Preferred	New	110‡
[Net->Listen]	Member	Preferred	New	110‡
[Net->LocalAddress]	Member	Preferred	New	109‡
[Net->ReadFrom]	Member	Preferred	New	114‡
[Net->Read]	Member	Preferred	New	110‡
[Net->RemoteAddress]	Member	Preferred	New	109‡
[Net->SetBlocking]	Member	Preferred	New	109‡
[Net->SetType]	Member	Preferred	New	109‡
[Net->Wait]	Member	Preferred	New	109‡
[Net->WriteTo]	Member	Preferred	New	114‡
[Net->Write]	Member	Preferred	New	110‡
[Net_ConnectInProgress]	Substitution	Preferred	New	108‡
[Net_ConnectOK]	Substitution	Preferred	New	108‡
[Net_WaitRead]	Substitution	Preferred	New	108‡
[Net_WaitTimeOut]	Substitution	Preferred	New	108‡
[Net_WaitWrite]	Substitution	Preferred	New	108‡

Operator

Tag	Type	Status	Change	Page
-Op	Command	Abbreviation	None	123*
[Op_LogicalValue]	Substitution	Synonym	None	120*
-OpBegin	Command	Abbreviation	None	123*
-OpEnd	Command	Abbreviation	None	123*
-Operator	Command	Preferred	None	123
[Operator_LogicalValue]	Substitution	Preferred	None	120
-OperatorBegin	Command	Preferred	None	123
-OperatorEnd	Command	Preferred	None	123
-OperatorLogical	Command	Preferred	None	123
-OpLogical	Command	Abbreviation	None	123*

Output

Tag	Type	Status	Change	Page
[Content_Type]	Substitution	Preferred	None	504
[File_Serve]	Process	Preferred	None	479
[Header]	Container	Preferred	None	504
[HTML_Comment]	Container	Preferred	None	247
[Output]	Substitution	Preferred	None	247
[Output_None]	Container	Preferred	None	247
[Redirect_URL]	Process	Preferred	None	487
[Server_Push]	Process	Preferred	None	503

PDF

Tag	Type	Status	Change	Page
[PDF_Barcode]	Substitution	Preferred	None	592
[PDF_Doc]	Substitution	Preferred	None	559
[PDF_Doc->Add]	Member	Preferred	New	565
[PDF_Doc->AddChapter]	Member	Preferred	None	574
[PDF_Doc->AddCheckBox]	Member	Preferred	None	577
[PDF_Doc->AddComboBox]	Member	Preferred	None	577
[PDF_Doc->AddHiddenField]	Member	Preferred	None	577
[PDF_Doc->AddPage]	Member	Preferred	None	574
[PDF_Doc->AddPasswordField]	Member	Preferred	None	577
[PDF_Doc->AddPhrase]	Member	Preferred	None	530
[PDF_Doc->AddRadioButton]	Member	Preferred	None	577
[PDF_Doc->AddRadioGroup]	Member	Preferred	None	577
[PDF_Doc->AddResetButton]	Member	Preferred	None	577
[PDF_Doc->AddSection]	Member	Preferred	None	574
[PDF_Doc->AddSelectList]	Member	Preferred	None	577
[PDF_Doc->AddSubmitButton]	Member	Preferred	None	577
[PDF_Doc->AddTextArea]	Member	Preferred	None	577
[PDF_Doc->AddTextField]	Member	Preferred	None	577
[PDF_Doc->Arc]	Member	Preferred	None	589
[PDF_Doc->Circle]	Member	Preferred	None	589
[PDF_Doc->ClosePath]	Member	Preferred	None	589
[PDF_Doc->Close]	Member	Preferred	None	567
[PDF_Doc->CurveTo]	Member	Preferred	None	589
[PDF_Doc->GetHeaders]	Member	Preferred	None	566
[PDF_Doc->GetMargins]	Member	Preferred	None	566
[PDF_Doc->GetPageNumber]	Member	Preferred	None	574
[PDF_Doc->GetSize]	Member	Preferred	None	566
[PDF_Doc->InsertPage]	Member	Preferred	New	563
[PDF_Doc->Line]	Member	Preferred	None	589
[PDF_Doc->MoveTo]	Member	Preferred	None	589
[PDF_Doc->Rect]	Member	Preferred	None	589
[PDF_Doc->SetColor]	Member	Preferred	None	589
[PDF_Doc->SetLineWidth]	Member	Preferred	None	589

[PDF_Doc->SetPageNumber]	Member	Preferred	None	574
[PDF_Font]	Substitution	Preferred	None	568
[PDF_Font->GetColor]	Member	Preferred	None	569
[PDF_Font->GetEncoding]	Member	Preferred	None	569
[PDF_Font->GetFace]	Member	Preferred	None	569
[PDF_Font->GetFullFontName]	Member	Preferred	None	569
[PDF_Font->GetPSFontName]	Member	Preferred	None	569
[PDF_Font->GetSize]	Member	Preferred	None	569
[PDF_Font->GetSupportedEncodings]	Member	Preferred	None	569
[PDF_Font->IsTrueType]	Member	Preferred	None	569
[PDF_Font->SetColor]	Member	Preferred	None	569
[PDF_Font->SetEncoding]	Member	Preferred	None	569
[PDF_Font->SetFace]	Member	Preferred	None	569
[PDF_Font->SetSize]	Member	Preferred	None	569
[PDF_Font->SeUnderline]	Member	Preferred	None	569
[PDF_Font->TextWidth]	Member	Preferred	New	569
[PDF_Image]	Substitution	Preferred	None	587
[PDF_List]	Substitution	Preferred	New	573
[PDF_List->Add]	Member	Preferred	New	573
[PDF_Read]	Substitution	Preferred	New	562
[PDF_Read->PageCount]	Member	Preferred	New	562
[PDF_Read->PageSize]	Member	Preferred	New	562
[PDF_Serve]	Process	Preferred	None	599
[PDF_Table]	Substitution	Preferred	None	583
[PDF_Table->Add]	Member	Preferred	New	583
[PDF_Table->GetAbsWidth]	Member	Preferred	None	584
[PDF_Table->GetColumnCount]	Member	Preferred	None	584
[PDF_Table->GetRowCount]	Member	Preferred	None	584
[PDF_Text]	Substitution	Preferred	New	571
[PDF_Text->Add]	Member	Preferred	New	571

Response

Tag	Type	Status	Change	Page
-Response	Command	Preferred	None	116
-ResponseAnyError	Command	Preferred	None	116
-ResponseReqColumnMissingError	Command	Abbreviation	None	116*
-ResponseReqFieldMissingError	Command	Abbreviation	None	116
-ResponseRequiredColumnMissingError	Command	Synonym	None	116*
-ResponseRequiredFieldMissingError	Command	Preferred	None	116*
-ResponseSecurityError	Command	Preferred	None	116

Results

Tag	Type	Status	Change	Page
[Found_Count]	Substitution	Preferred	None	125
[Lasso_CurrentAction]	Substitution	Preferred	None	120
[MaxRecords_Value]	Substitution	Preferred	None	120
[Req_Column]	Substitution	Abbreviation	None	127*
[Req_Field]	Substitution	Abbreviation	None	127*
[Required_Column]	Substitution	Synonym	None	127*
[Required_Field]	Substitution	Preferred	None	127
[Response_FilePath]	Substitution	Preferred	None	507
[Response_LocalPath]	Substitution	Preferred	None	507
[Response_Path]	Substitution	Preferred	None	507
[Response_Realm]	Substitution	Preferred	None	507
[Search_Args]	Container	Abbreviation	None	120*
[Search_Arguments]	Container	Preferred	None	120
[Search_ColumnItem]	Substitution	Synonym	None	120*
[Search_FieldItem]	Substitution	Preferred	None	120
[Search_OperatorItem]	Substitution	Preferred	None	120
[Search_Opltem]	Substitution	Abbreviation	None	120*
[Search_Valueltem]	Substitution	Preferred	None	120
[Shown_Count]	Substitution	Preferred	None	125
[Shown_First]	Substitution	Preferred	None	125
[Shown_Last]	Substitution	Preferred	None	125
[SkipRecords_Value]	Substitution	Preferred	None	120
[Sort_Args]	Substitution	Abbreviation	None	120*
[Sort_Arguments]	Container	Preferred	None	120
[Sort_ColumnItem]	Substitution	Synonym	None	120*
[Sort_FieldItem]	Substitution	Preferred	None	120
[Sort_OrderItem]	Substitution	Preferred	None	120
[Total_Records]	Substitution	Preferred	None	125

Session

Tag	Type	Status	Change	Page
[Session_Abort]	Process	Preferred	New	375
[Session_AddVar]	Process	Abbreviation	None	375*
[Session_AddVariable]	Process	Preferred	None	375
[Session_End]	Process	Preferred	None	375
[Session_ID]	Substitution	Preferred	None	375
[Session_RemoveVar]	Process	Abbreviation	None	375*
[Session_RemoveVariable]	Process	Preferred	None	375
[Session_Result]	Substitution	Preferred	New	375
[Session_Start]	Process	Preferred	None	375

String

Tag	Type	Status	Change	Page
[String]	Substitution	Preferred	None	281
[String->Append]	Member	Preferred	None	286
[String->BeginsWith]	Member	Preferred	None	290
[String->CharDigitValue]	Member	Preferred	New	295
[String->CharName]	Member	Preferred	New	295
[String_CharFromName]	Substitution	Preferred	New	297
[String->CharType]	Member	Preferred	New	295
[String->CompareCodePointOrder]	Member	Preferred	New	290*
[String->Compare]	Member	Preferred	New	290
[String_Concatenate]	Substitution	Preferred	None	287
[String->Contains]	Member	Preferred	None	290
[String->Digit]	Member	Preferred	New	295
[String->EndsWith]	Member	Preferred	None	290
[String_EndsWith]	Substitution	Preferred	None	291
[String->Equals]	Member	Preferred	None	290
[String_Extract]	Substitution	Preferred	None	293
[String->Find]	Member	Preferred	None	292
[String_FindPosition]	Substitution	Preferred	None	293
[String_FindRegExp]	Substitution	Preferred	None	298
[String->FoldCase]	Member	Preferred	New	289
[String->Get]	Member	Preferred	None	292
[String->GetNumericValue]	Member	Preferred	New	295
[String_GetUnicodeVersion]	Substitution	Preferred	New	297
[String_Insert]	Substitution	Preferred	None	287
[String->IsAlnum]	Member	Preferred	New	295
[String->IsAlpha]	Member	Preferred	New	295
[String_IsAlpha]	Substitution	Preferred	None	293
[String_IsAlphaNumeric]	Substitution	Preferred	None	293
[String->IsBase]	Member	Preferred	New	295
[String->IsCntrl]	Member	Preferred	New	295
[String->IsDefined]	Member	Preferred	New	295
[String->IsDigit]	Member	Preferred	New	295
[String_IsDigit]	Substitution	Preferred	None	293
[String_IsHexdigit]	Substitution	Preferred	None	293
[String->IsLower]	Member	Preferred	New	295
[String_IsLower]	Substitution	Preferred	None	293
[String_IsNumeric]	Substitution	Preferred	None	293
[String->IsPrint]	Member	Preferred	New	295
[String_IsPunctuation]	Substitution	Preferred	None	293
[String->IsSpace]	Member	Preferred	New	295
[String_IsSpace]	Substitution	Preferred	None	293
[String->IsTitle]	Member	Preferred	New	295
[String->IsUAlphabetic]	Member	Preferred	New	295
[String->IsULowerCase]	Member	Preferred	New	295
[String->IsUpper]	Member	Preferred	New	295

[String_IsUpper]	Substitution	Preferred	None	293
[String->isUpperCase]	Member	Preferred	New	295
[String->IsWhiteSpace]	Member	Preferred	New	295
[String->IsWhiteSpace]	Member	Preferred	New	295
[String_Length]	Substitution	Preferred	None	293
[String->LowerCase]	Member	Preferred	None	289
[String_LowerCase]	Substitution	Preferred	None	289
[String->Merge]	Member	Preferred	None	286
[String->PadLeading]	Member	Preferred	New	286
[String->PadTrailing]	Member	Preferred	New	286
[String->Remove]	Member	Preferred	None	286
[String_Remove]	Substitution	Preferred	None	287
[String->RemoveLeading]	Member	Preferred	None	286
[String_RemoveLeading]	Substitution	Preferred	None	287
[String->RemoveTrailing]	Member	Preferred	None	286
[String_RemoveTrailing]	Substitution	Preferred	None	287
[String->Replace]	Member	Preferred	None	286
[String_Replace]	Substitution	Preferred	None	287
[String_ReplaceRegExp]	Substitution	Preferred	None	298
[String->Reverse]	Member	Preferred	New	286
[String->Size]	Member	Preferred	None	292
[String->Split]	Member	Preferred	None	297
[String->Substring]	Member	Preferred	None	292
[String->TitleCase]	Member	Preferred	New	289
[String->ToLower]	Member	Preferred	New	289
[String->ToTitle]	Member	Preferred	New	289
[String->ToUpper]	Member	Preferred	New	289
[String->Trim]	Member	Preferred	None	286
[String->Unescape]	Member	Preferred	New	289
[String->Uppercase]	Member	Preferred	None	289
[String_Uppercase]	Substitution	Preferred	None	289

Technical

Tag	Type	Status	Change	Page
[Boolean]	Substitution	Preferred	None	275
[Bytes]	Substitution	Preferred	New	92‡
[Bytes->Append]	Member	Preferred	New	92‡
[Bytes->BeginsWith]	Member	Preferred	New	92‡
[Bytes->Contains]	Member	Preferred	New	92‡
[Bytes->EndsWith]	Member	Preferred	New	92‡
[Bytes->Export16Bits]	Member	Preferred	New	92‡
[Bytes->Export32Bits]	Member	Preferred	New	92‡
[Bytes->Export64Bits]	Member	Preferred	New	92‡
[Bytes->Export8Bits]	Member	Preferred	New	92‡
[Bytes->ExportString]	Member	Preferred	New	92‡
[Bytes->Find]	Member	Preferred	New	92‡
[Bytes->GetRange]	Member	Preferred	New	92‡

[Bytes->Get]	Member	Preferred	New	92‡
[Bytes->Import16Bits]	Member	Preferred	New	92‡
[Bytes->Import32Bits]	Member	Preferred	New	92‡
[Bytes->Import64Bits]	Member	Preferred	New	92‡
[Bytes->Import8Bits]	Member	Preferred	New	92‡
[Bytes->ImportString]	Member	Preferred	New	92‡
[Bytes->Length]	Member	Preferred	New	92‡
[Bytes->RemoveLeading]	Member	Preferred	New	92‡
[Bytes->RemoveTrailing]	Member	Preferred	New	92‡
[Bytes->Replace]	Member	Preferred	New	92‡
[Bytes->SetRange]	Member	Preferred	New	92‡
[Bytes->SetSize]	Member	Preferred	New	92‡
[Bytes->Size]	Member	Preferred	New	92‡
[Bytes->Split]	Member	Preferred	New	92‡
[Bytes->SwapBytes]	Member	Preferred	New	92‡
[Bytes->Trim]	Member	Preferred	New	92‡
[Compress]	Substitution	Preferred	None	371
[Decompress]	Substitution	Preferred	None	371
[Lasso_ExecutionTimeLimit]	Process	Preferred	New	442
[LassoApp_Create]	Substitution	Preferred	None	15‡
[LassoApp_Dump]	Substitution	Preferred	None	15‡
[LassoApp_Link]	Substitution	Preferred	None	15‡
[LassoApp_List]	Substitution	Preferred	New	15‡
[NoProcess]	Container	Preferred	None	435
[Null]	Substitution	Preferred	None	437
[Null->DetachReference]	Member	Preferred	None	437
[Null->Dump]	Member	Deprecated	None	437
[Null->FreezeType]	Member	Preferred	None	437
[Null->FreezeValue]	Member	Preferred	None	437
[Null->IsA]	Member	Preferred	None	437
[Null->Parent]	Member	Preferred	None	437
[Null->Properties]	Member	Preferred	None	437
[Null->RefCount]	Member	Preferred	None	437
[Null->Serialize]	Member	Preferred	None	437
[Null->Type]	Member	Preferred	None	437
[Null->Unserialize]	Member	Preferred	None	437
[Null->Up]	Member	Deprecated	None	437
[Null->XMLSchemaType]	Member	Preferred	New	437
[Process]	Substitution	Preferred	None	435
[Reference]	Substitution	Preferred	None	88‡
-ResponseLassoApp	Command	Preferred	None	15‡
[Sleep]	Process	Preferred	None	435
[Tag]	Substitution	Preferred	None	95‡
[Tag->AsAsync]	Member	Preferred	None	95‡
[Tag->AsType]	Member	Preferred	None	95‡
[Tag->Description]	Member	Preferred	None	95‡
[Tag->Eval]	Member	Preferred	None	95‡
[Tag->Run]	Member	Preferred	None	95‡

[Thread_Event]	Substitution	Preferred	None	104‡
[Thread_Event->SignalAll]	Member	Preferred	None	104‡
[Thread_Event->Signal]	Member	Preferred	None	104‡
[Thread_Event->Wait]	Member	Preferred	None	104‡
[Thread_Lock]	Substitution	Preferred	None	100‡
[Thread_Lock->Lock]	Member	Preferred	None	100‡
[Thread_Lock->UnLock]	Member	Preferred	None	100‡
[Thread_Pipe]	Substitution	Preferred	None	104‡
[Thread_Pipe->Get]	Member	Preferred	None	105‡
[Thread_Pipe->Set]	Member	Preferred	None	105‡
[Thread_RWLock]	Substitution	Preferred	None	100‡
[Thread_RWLock->ReadLock]	Member	Preferred	None	103‡
[Thread_RWLock->ReadUnlock]	Member	Preferred	None	103‡
[Thread_RWLock->WriteLock]	Member	Preferred	None	103‡
[Thread_RWLock->WriteUnlock]	Member	Preferred	None	103‡
[Thread_Semaphore]	Substitution	Preferred	None	100‡
[Thread_Semaphore->Decrement]	Member	Preferred	None	102‡
[Thread_Semaphore->Increment]	Member	Preferred	None	102‡

Utility

Tag	Type	Status	Change	Page
[Action_Param]	Substitution	Preferred	None	120
[Action_Params]	Substitution	Preferred	None	120
[Cache]	Container	Preferred	New	497
[Cache_Empty]	Process	Preferred	New	501
[Cache_Fetch]	Substitution	Preferred	New	501
[Cache_Object]	Substitution	Preferred	New	500
[Cache_Store]	Process	Preferred	New	500
[Cookie]	Substitution	Preferred	None	492
[Cookie_Set]	Process	Preferred	None	492
[Email_Send]	Process	Preferred	None	456
[Event_Schedule]	Process	Preferred	None	432
[FTP_GetFile]	Substitution	Preferred	None	489
[FTP_GetListing]	Substitution	Preferred	None	489
[FTP_PutFile]	Process	Preferred	None	489
[HTTP_GetFile]	Substitution	Preferred	None	488
[Inline]	Container	Preferred	None	110
[Lasso_UniqueID]	Substitution	Preferred	None	445
[Log]	Container	Preferred	None	289
[Log_Critical]	Process	Preferred	None	390
[Log_Detail]	Process	Preferred	None	390
[Log_Warning]	Process	Preferred	None	390
[NSLookup]	Substitution	Preferred	None	443
-Password	Command	Preferred	None	139
[Server_Name]	Substitution	Preferred	None	509
[Server_Port]	Substitution	Preferred	None	509
-Token	Command	Preferred	None	123

[Token_Value]	Substitution	Preferred	None	120
-Username	Command	Preferred	None	139
[Valid_CreditCard]	Substitution	Preferred	None	444
[Valid_Date]	Substitution	Preferred	New	444
[Valid_Email]	Substitution	Preferred	None	444
[Valid_URL]	Substitution	Preferred	None	444

Variable

Tag	Type	Status	Change	Page
[Global]	Substitution	Preferred	None	90‡
[Global_Defined]	Substitution	Preferred	None	90‡
[Globals]	Substitution	Preferred	None	90‡
[Local]	Substitution	Preferred	None	33‡
[Local_Defined]	Substitution	Preferred	None	33‡
[Locals]	Substitution	Preferred	None	33‡
[Var]	Substitution	Abbreviation	None	250
[Var_Defined]	Substitution	Abbreviation	None	250
[Var_Remove]	Process	Preferred	New	250
[Variable]	Substitution	Preferred	None	250
[Variable_Defined]	Substitution	Preferred	None	250
[Variables]	Substitution	Preferred	None	439
[Vars]	Substitution	Abbreviation	None	439*

XML

Tag	Type	Status	Change	Page
[XML]	Substitution	Preferred	None	522
[XML->Attributes]	Member	Preferred	None	523
[XML->Children]	Member	Preferred	None	523
[XML->Contents]	Member	Preferred	None	523
[XML->Document]	Member	Preferred	New	523
[XML->ExtractOne]	Member	Preferred	New	523
[XML->Extract]	Member	Preferred	New	523
[XML->Name]	Member	Preferred	None	523
[XML->Namespaces]	Member	Preferred	New	523
[XML->NextSibling]	Member	Preferred	New	523
[XML->Parent]	Member	Preferred	New	523
[XML->PreviousSibling]	Member	Preferred	New	523
[XML->Transform]	Member	Preferred	New	523
[XML_Extract]	Substitution	Preferred	None	526
[XML_Serve]	Process	Preferred	None	550
[XML_Transform]	Substitution	Preferred	None	531
[XML_RPC]	Substitution	Preferred	None	540
[XML_RPCCall]	Substitution	Preferred	None	538
[XML_RPC->Call]	Member	Preferred	None	540
[XML_RPC->GetMethod]	Member	Preferred	None	540
[XML_RPC->GetParams]	Member	Preferred	None	540

[XML_RPC->Response]	Member	Preferred	None	540
[XMLStream]	Substitution	Preferred	New	532
[XMLStream->AttributeCount]	Member	Preferred	New	535
[XMLStream->BaseURI]	Member	Preferred	New	535
[XMLStream->Depth]	Member	Preferred	New	535
[XMLStream->GetAttributeNamespace]	Member	Preferred	New	535
[XMLStream->GetAttribute]	Member	Preferred	New	535
[XMLStream->HasAttributes]	Member	Preferred	New	535
[XMLStream->HasValue]	Member	Preferred	New	535
[XMLStream->IsEmptyElement]	Member	Preferred	New	535
[XMLStream->LocalName]	Member	Preferred	New	535
[XMLStream->LookupNamespace]	Member	Preferred	New	535
[XMLStream->MoveToAttributeNamespace]	Member	Preferred	New	534
[XMLStream->MoveToElement]	Member	Preferred	New	534
[XMLStream->MoveToFirstAttribute]	Member	Preferred	New	534
[XMLStream->MoveToNextAttribute]	Member	Preferred	New	534
[XMLStream->Name]	Member	Preferred	New	535
[XMLStream->NamespaceURI]	Member	Preferred	New	535
[XMLStream->NextSibling]	Member	Preferred	New	534
[XMLStream->Next]	Member	Preferred	New	534
[XMLStream->NodeType]	Member	Preferred	New	535
[XMLStream->Prefix]	Member	Preferred	New	535
[XMLStream->ReadAttributeValue]	Member	Preferred	New	535
[XMLStream->ReadString]	Member	Preferred	New	535
[XMLStream->Value]	Member	Preferred	New	535
[XMLStream->XMLLang]	Member	Preferred	New	535

LDML 7 Legacy Tag List

All tags in this list have been deprecated. This means that they are supported in Lasso Professional 7, but may not be supported in a future version of Lasso. Equivalent tags are LDML 7 preferred tags which provide the same basic functionality as associated legacy tags, but may not share exactly the same syntax. While not required, for future compatibility it is recommended that solutions containing legacy tags be Noned using the equivalent preferred tags. Additional information is available in the upgrading chapters of the documentation.

Administration

Tag	Type	Equivalent
[Lasso_DataType]	Substitution	[Null->Type]

Client

Tag	Type	Equivalent
[Client_Addr]	Substitution	[Client_Address]

Conditional

Tag	Type	Equivalent
[Lasso_Abort]	Process	[Abort]
[Repetition]	Substitution	(Loop_Count % 2) == 0

Custom Tag

Tag	Type	Equivalent
[Named_Param]	Substitution	[Define_Tag: -Required] or [Define_Tag: -Optional]

Database

Tag	Type	Equivalent
[Choice_List]	Container	[Value_List]
[ChoiceListItem]	Container	[Value_ListItem]
-Datasource	Command	-Database
[Datasource_Name]	Substitution	[Database_Name]
[DB_LayoutNameItem]	Substitution	[Database_TableNameItem]
[DB_LayoutNames]	Container	[Database_TableNames]
[DB_NameItem]	Substitution	[Database_NameItem]
[DB_Names]	Substitution	[Database_Names]
[DB_TableNameItem]	Substitution	[Database_TableNameItem]
[DB_TableNames]	Container	[Database_TableNames]
-DoScript	Command	-FMScript
-DoScript.Post	Command	-FMScriptPost
-DoScript.Pre	Command	-FMScriptPre
-DoScript.PreSort	Command	-FMScriptPreSort
-RecID	Command	-KeyValue
[RecID_Value]	Substitution	[KeyField_Value]
-RecordID	Command	-KeyValue
[RecordID_Value]	Substitution	[KeyField_Value]
[RepeatingValueItem]	Command	[Repeating_ValueItem]
[ValueListItem]	Substitution	[Value_ListItem]

Date

Tag	Type	Equivalent
[Date_GetCurrentDate]	Substitution	[Date]
[Date_GetDay]	Substitution	[Date->Day]
[Date_GetDayOfWeek]	Substitution	[Date->DayOfWeek]
[Date_GetHour]	Substitution	[Date->Hour]
[Date_GetMinute]	Substitution	[Date->Minute]
[Date_GetMonth]	Substitution	[Date->Month]
[Date_GetSecond]	Substitution	[Date->Second]
[Date_GetTime]	Substitution	[Date->Time]
[Date_GetYear]	Substitution	[Date->Year]

Encoding

Tag	Type	Equivalent
[Encode_Breaks]	Substitution	[Encode_Break]
[Encode_Raw]	Substitution	[Encode_None]

Error

Tag	Type	Equivalent
[Error_NoRecordsFound]	Substitution	[Found_Count]

File

Tag	Type	Equivalent
[Directory_Lister]	Container	[File_ListDirectory]
[DirectoryNameItem]	Substitution	[File_ListDirectory]
[File_Control]	Container	None

Include

Tag	Type	Equivalent
[Include_CGI]	Container	[Include_URL]

Link

Tag	Type	Equivalent
[Link_CurrentSearch]	Container	[Link_CurrentAction]
[Link_CurrentSearchURL]	Substitution	[Link_CurrentActionURL]
[Shown_NextGroup]	Container	[Link_NextGroup]
[Shown_NextGroupURL]	Substitution	[Link_NextGroupURL]
[Shown_PrevGroup]	Container	[Link_PrevGroup]
[Shown_PrevGroupURL]	Substitution	[Link_PrevGroupURL]

List

Tag	Type	Equivalent
[List_AddItem]	Substitution	[Array->Insert]
[List_FromList]	Substitution	[Array]
[List_FromString]	Substitution	[String->Split]
[List_GetItem]	Substitution	[Array->Get]
[List_ItemCount]	Substitution	[Array->Size]
[List_RemoveItem]	Process	[Array->Remove]
[List_ReplaceItem]	Process	[Array->Get]
[List_ToString]	Substitution	[Array->Get]

Math

Tag	Type	Equivalent
[Euro]	Substitution	[Math_ConvertEuro]
[RandomNumber]	Substitution	[Math_Random]
[Roman]	Substitution	[Math_Roman]

Operator

Tag	Type	Equivalent
-LogicalOp	Command	-OperatorLogical
-LogicalOperator	Command	-OperatorLogical
[LogicalOp_Value]	Substitution	[Operator_LogicalValue]
[LogicalOperator_Value]	Substitution	[Operator_LogicalValue]

Output

Tag	Type	Equivalent
[Lasso_Comment]	Container	[Output_None]

Response

Tag	Type	Equivalent
-AddError	Command	-ResponseAnyError
-AddResponse	Command	-Response
-AnyError	Command	-ResponseAnyError
-AnyResponse	Command	-Response
-DeleteResponse	Command	-Response
-DuplicateResponse	Command	-Response
-NoResultsError	Command	-ResponseAnyError
-RequiredColumnMissingError	Command	-ResponseAnyError
-RequiredFieldMissingError	Command	-ResponseAnyError
-SecurityError	Command	-ResponseSecurityError
-NoneError	Command	-ResponseAnyError
-NoneResponse	Command	-Response
-ResponseAdd	Command	-Response
-ResponseAddError	Command	-Response
-ResponseAny	Command	-Response
-ResponseDelete	Command	-Response
-ResponseDuplicate	Command	-Response
-ResponseNoResultsError	Command	-Response
-ResponseUpdate	Command	-Response
-ResponseUpdateError	Command	-Response

Results

Tag	Type	Equivalent
[Record_Count]	Substitution	[Shown_Count]
[Row_Count]	Substitution	[Shown_Count]
[SearchFieldItem]	Substitution	[Search_FieldItem]
[SearchOperatorItem]	Substitution	[Search_OperatorItem]
[SearchOpItem]	Substitution	[Search_OperatorItem]
[SearchValueItem]	Substitution	[Search_ValueItem]
[SortColumnItem]	Substitution	[Sort_FieldItem]
[SortFieldItem]	Substitution	[Sort_FieldItem]
[SortOrderItem]	Substitution	[Sort_OrderItem]

String

Tag	Type	Equivalent
[String_CountFields]	Substitution	[Array->Size]
[String_GetField]	Substitution	[Array->Get]
[String->Length]	Member	[String->Size]
[String_ToDecimal]	Substitution	[Decimal]
[String_ToInteger]	Substitution	[Integer]

Technical

Tag	Type	Equivalent
[Lasso_Process]	Substitution	[Process]

Utility

Tag	Type	Equivalent
-ClientPassword	Command	-Password
-ClientUsername	Command	-Username
[Form_Param]	Substitution	[Action_Param]
[Lasso_SessionID]	Substitution	[Lasso_UniqueID]
[SQL_Inline]	Container	[Inline: -SQL]
[TCP_Close]	Substitution	[Net]
[TCP_Open]	Substitution	[Net]
[TCP_Send]	Substitution	[Net]

B

Appendix B

Error Codes

This appendix contains a list of all known error codes that Lasso Professional 7, Lasso MySQL, or FileMaker Pro will return.

- *Lasso Professional 7 Error Codes* contains a list of all error codes which are generated by Lasso Professional 7.
- *Lasso MySQL Error Codes* contains a list of all error codes which are generated by Lasso MySQL or another MySQL data source.
- *FileMaker Pro Error Codes* contains a list of known error codes which are generated by FileMaker Pro when used as a data source.

In addition to the error codes described in this appendix, Lasso Professional 7 will report any unknown errors it receives from the operating system, Web server applications, or data source applications it communicates with. Please consult the documentation for the operating system and each application for more information about the error codes they may report.

For information about how to gracefully handle and recover from errors, please see *Chapter 21: Error Control*.

Lasso Professional 7 Error Codes

The following *Table 1: Lasso Professional Error Codes* lists all of the native error codes of Lasso Professional 7. The error codes are listed in numerical order and are divided into general categories for easier reading. Many of the error codes descriptions contain helpful information about what to do to correct or prevent the error.

Table 1: Lasso Professional 7 Error Codes

Error Code	Description
0	No Error.
-609	The specified database was not found. Lasso could not find the specified database. This error usually occurs when a database is not open or not accessible by Lasso. Make sure the specified database is open.
-700	Could not find email format file. The format file specified by an -Email.Format command tag could not be found. Check the spelling of the file name. Make sure the path to the file is specified properly.
-701	All email tags must be assigned a value. In order for an email message to be sent, all five of the email parameters (-Email.Host, -Email.From, -Email.To, -Email.Subject, and -Email.Format) must be specified. Make sure you have specified values for all five parameters in your HTML form. Make sure the parameter names are spelled correctly.
Database Errors	
-800	Value missing for required field. The value of one or more required field was not specified. Make sure that all required fields are supplied with a value.
-801	Repeating related fields are not supported. An attempt to retrieve data from a repeating related field failed. Lasso does not support retrieving data from repeating related fields.
-802	Action not supported. The specified Lasso action is not supported by the specified database or data source.
-1712	Timeout. A database action timed out.
-1728	No records found. No records were found in the specified database.
-2000	The module was not found. The requested module was not found. Make sure that the module is located in the "Lasso Modules" folder and relaunch the Web server and/or Lasso.
-3000	A data source error has occurred.

continued

Syntax Errors

-9951	A syntax error occurred. Invalid or incorrect syntax was used. Correct the syntax.
-9952	A looping tag was aborted.
-9953	Unknown error.

Internal Errors

-9954	A pointer was nil when it should not have been.
-9955	Overflow: Some memory passed to a function that was too small to hold the results.
-9956	An invalid parameter was passed to a function.

Action Errors

-9957	Delete error. An error occurred while deleting a record from the specified database. Make sure that the database or data source is set to allow record deletion.
-9958	Update error. An error occurred while updating a record from the specified database. Make sure that the database or data source is set to allow records to be updated.
-9959	Add error. An error occurred while adding a record to the specified database. Make sure that the database or data source is set to allow records to be added.
-9960	Field restriction. A field security restriction prevented the action from being executed. Edit field security restrictions as configured within Lasso security.

Security Errors

-9961	No permission. The current user is not allowed to perform the specified action. This could mean that a file suffix is not allowed by Lasso security. Edit user security permissions as configured within Lasso security.
-9962	Invalid database. The database or data source name is not valid.
-9963	Invalid password. The password supplied is not valid.
-9964	Invalid user name. The user name supplied is not valid.
-9965	Network error. An error occurred accessing the network connection. This error usually occurs while communicating with FileMaker Pro over TCP/IP. Try quitting and restarting the FileMaker Pro client.
-9966	Resource error.
-9967	Resource not found.

continued

File Errors

-9968	Could not read from file.
-9969	Could not write to file.
-9970	End of file reached.
-9971	Beginning of file reached.
-9972	File is closed.
-9973	File already open with write permission.
-9974	File Unlocked.
-9975	File locked.
-9976	Invalid filename.
-9977	Invalid pathname.
-9978	I/O error.
-9979	Directory full.
-9980	Disk full.
-9981	Volume does not exist.
-9982	The file is corrupt.
-9983	File already exists.
-9984	Unauthorized file suffix or file not found. The error -9984 can be seen if you specify a format file with a file suffix which is not included in the Lasso Security settings. Also returned by file management tags.
-9985	Could not delete file.
-9986	Could not close file.
-9987	Could not create or open file.
-9988	Invalid access mode.
-9990	File error.

continued

Memory Errors

-9991	Could not dispose memory.
-9992	Could not unlock memory.
-9993	Could not lock memory.
-9994	Lasso ran out of stack space. This error may occur when a Lasso format file contains too many deeply nested container tags. The [Variable] tag can be used in order to significantly reduce the number on nested tags in a format file.
-9995	Lasso ran out of memory. Increase the memory which is available to the server running Lasso.
-9996	Invalid memory object.
-9997	Memory error.
-9998	Error writing to stream.
-9999	Error reading from stream.

Lasso MySQL Error Codes

All of the known error codes in Lasso MySQL are listed in *Table 2: Lasso MySQL Error Codes*. Additional error codes may be reported if Lasso MySQL encounters an operating system error. If Lasso receives one of these error codes from Lasso MySQL or another MySQL data source then it will be passed on to the site visitor.

Table 2: Lasso MySQL Error Codes

Error Code	Description
1	Operation not permitted.
2	No such file or directory.
3	No such process.
4	Interrupted system call.
5	Input/output error.
6	Device not configured.
7	Argument list too long.
8	Exec format error.
9	Bad file descriptor.
10	No child processes.
11	Resource deadlock avoided.
12	Cannot allocate memory.
13	Permission denied.
14	Bad address.
15	Block device required.
16	Device busy.
17	File exists.
18	Cross-device link.
19	Operation not supported by device.
20	Not a directory.
21	Is a directory.
22	Invalid argument.
23	Too many open files in system.
24	Too many open files.
25	Inappropriate ioctl for device.

continued

26 – 50

26	Text file busy.
27	File too large.
28	No space left on device.
29	Illegal seek.
30	Read-only file system.
31	Too many links.
32	Broken pipe.
33	Numerical argument out of domain.
34	Result too large.
35	Resource temporarily unavailable.
36	Operation now in progress.
37	Operation already in progress.
38	Socket operation on non-socket.
39	Destination address required.
40	Message too long.
41	Protocol wrong type for socket.
42	Protocol not available.
43	Protocol not supported.
44	Socket type not supported.
45	Operation not supported.
46	Protocol family not supported.
47	Address family not supported by protocol family.
48	Address already in use.
49	Can't assign requested address.
50	Network is down.

continued

51 – 75

51	Network is unreachable.
52	Network dropped connection on reset.
53	Software caused connection abort.
54	Connection reset by peer.
55	No buffer space available.
56	Socket is already connected.
57	Socket is not connected.
58	Can't send after socket shutdown.
59	Too many references: can't splice.
60	Operation timed out.
61	Connection refused.
62	Too many levels of symbolic links.
63	File name too long.
64	Host is down.
65	No route to host.
66	Directory not empty.
67	Too many processes.
68	Too many users.
69	Disc quota exceeded.
70	Stale NFS file handle.
71	Too many levels of remote in path.
72	RPC struct is bad.
73	RPC version wrong.
74	RPC prog. not avail.
75	Program version wrong.

continued

76 – 150

76	Bad procedure for program.
77	No locks available.
78	Function not implemented.
79	Inappropriate file type or format.
80	Authentication error.
81	Need authenticator.
82	Device power is off.
83	Device error.
84	Value too large to be stored in data type.
85	Bad executable (or shared library).
86	Bad CPU type in executable.
87	Shared library version mismatch.
88	Malformed Mach-O library file.
120	Didn't find key on read or update.
121	Duplicate key on write or update.
123	Someone has changed the row since it was read.
124	Wrong index given to function.
126	Index file is crashed / Wrong file format.
127	Record-file is crashed.
131	Command not supported by database.
132	Old database file.
133	No record read before update.
134	Record was already deleted (or record file crashed).
135	No more room in record file.
136	No more room in index file.
137	No more records (read after end of file).
138	Unsupported extension used for table.
139	Too big row (≥ 16 M).
140	Wrong create options.
141	Duplicate unique key or constraint on write or update.
142	Unknown character set used.
143	Conflicting table definition between MERGE and mapped table.
144	Table is crashed and last repair failed.
145	Table was marked as crashed and should be repaired.

FileMaker Pro Error Codes

All of the known error codes for the FileMaker Pro Web Companion as of FileMaker Pro 5.5v3 are listed in *Table 3: FileMaker Pro Error Codes*. Additional error codes may be reported if FileMaker Pro encounters an operating system error. If Lasso receives one of these error codes from a FileMaker Pro data source, it will be passed on to the site visitor.

Table 3: FileMaker Pro Error Codes

Error Code	Description
-1	Unknown Error.
0	No Error.
1	User cancelled action.
2	Memory error.
3	Command is unavailable.
4	Command is unknown.
5	Command is invalid.
100 – 199	
100	File is missing.
101	Record is missing.
102	Field is missing.
103	Relation is missing.
104	Script is missing.
105	Layout is missing.
200 – 299	
200	Record access is denied.
201	Field cannot be modified.
202	Field access is denied.
203	No records in file to print or password doesn't allow print access.
204	No access to field(s) in sort order.
205	Cannot create new records; import will overwrite existing data.

continued

300 – 399

300	The file is locked or in use.
301	Record is in use by another user.
302	Script definitions are in use by another user.
303	Paper size is in use by another user.
304	Password definitions are in use by another user.
305	Relationship or value list definitions are locked by another user.

400 – 499

400	Find criteria is empty.
401	No records match the request.
402	Not a match field for a lookup.
403	Exceeding maximum record limit for demo.
404	Sort order is invalid.
405	Number of records specified exceeds number of records that can be omitted.
406	Replace/Reserialize criteria is invalid.
407	One or both key fields are missing (invalid relation).
408	Specified field has inappropriate data type for this operation.
409	Import order is invalid.
410	Export order is invalid.
411	Cannot perform delete because related records cannot be deleted.
412	Wrong version of FileMaker Pro used to recover file.

continued

500 – 599

500	Date value does not meet validation entry options.
501	Time value does not meet validation entry options.
502	Number value does not meet validation entry options.
503	Value in field does not meet range validation entry options.
504	Value in field does not meet unique value validation entry options.
505	Value in field failed existing value validation test.
506	Value in field is not a member value of the validation entry option value list.
507	Value in field failed calculation test of validation entry option.
508	Value in field failed query value test of validation entry option.
509	Field requires a valid value.
510	Related value is empty or unavailable.

600 – 699

600	Print error has occurred.
601	Combined header and footer exceed one page .
602	Body doesn't fit on a page for current column setup .
603	Print connection lost.

700 – 799

700	File is of the wrong file type for import.
701	Data Access Manager can't find database extension file.
702	Data Access Manager was unable to open the session.
704	Data Access Manager failed when sending a query.
705	Data Access Manager failed when executing a query.
706	EPSF file has no preview image.
707	Graphic translator can not be found.
708	Can't import the file or need color computer.
709	QuickTime movie import failed.
710	Unable to update Quicktime file reference, read-only.
711	Import Translator can not be found.
712	XTND version is incompatible.
713	Couldn't initialize the XTND system.
714	Insufficient password privileges to allow the operation.

continued

800 – 899

800	Unable to create file on disk.
801	Unable to create temporary file on System disk.
802	Unable to open file .
803	File is single user or host cannot be found .
804	File cannot be opened as read-only in its current state .
805	File is damaged; use Recover command.
806	File cannot be opened with this version of FileMaker Pro.
807	File is not a FileMaker Pro file or is severely damaged .
808	Cannot open file because of damaged access privileges
809	Disk/volume is full.
810	Disk/volume is locked.
811	Temporary file cannot be opened as FileMaker Pro file.
812	Cannot open the file because it exceeds host capacity.
813	Record Synchronization error on network.
814	File(s) cannot be opened because maximum number is open.
815	Couldn't open lookup file.
816	Unable to convert file.

900 – 999

900	General spelling engine error.
901	Main spelling dictionary not installed.
902	Could not launch the Help system.
903	Command cannot be used in a shared file.
904	Command can only be used in a file hosted under FileMaker Server.
950	Adding repeating related fields is not supported.
951	An unexpected error occurred.
971	The user name is invalid.
972	The password is invalid.
973	The database is invalid.
974	Permission denied.
975	The field has restricted access.
976	Security is disabled.
977	Invalid client IP address (FileMaker Pro 5.x only).
978	The number of allowed guests has been exceeded (FileMaker Pro 5.x only).

JDBC Error Codes

All error codes specific to JDBC data sources are listed in *Table 4: JDBC Error Codes*. Additional error codes may be reported if the JDBC data source encounters an operating system error. If Lasso receives one of these error codes from a JDBC data source, it will be passed on to the site visitor.

Table 4: JDBC Error Codes

Error Code	Description
-11000	Invalid Token Error. Invalid Lasso state token passed from Java.
-10999	Null Parameter Error. One of the required parameters was Null.

Appendix C

Index

SYMBOLS

!	Boolean not 276		
!=	Boolean inequality 276 Mathematical inequality 310 String inequality 283	+	Mathematical multiplication 309 String repetition 283
"	HTML delimiter 88		
#	Local variable 250 URL delimiter 88		
\$	Page variable 250 Regular expressions 299		
%	Mathematical modulus 308	+=	String concatenation 283
%=	Mathematical modulus 309		
&	URL delimiter 88	,	Tag delimiter 87
&&	Boolean and 275	-	Date Subtraction 337 Keyword prefix 87 Mathematical subtraction 308 String deletion 282
,	String delimiter 87	-=	Mathematical subtraction 309 String deletion 283
()	Expression delimiter 87 Regular expressions 300	->	Member symbol 87
*	Mathematical multiplication 308 Regular expressions 300 String repetition 282	-Schema	240
*=		.	Regular expressions 299
		/	Mathematical division 308 URL delimiter 88
		//	LassoScript 448 LassoScript comment 87
		/=	Mathematical division 309
		:	Naming related fields 219

- Tag delimiter 87
- ;
- LassoScript delimiter 448
- Tag delimiter 87
- <
 - HTML delimiter 88
 - Mathematical less than 310
 - String order 283
- <=
 - Mathematical less than or equal 310
 - String order 283
- <?LassoScript 448
- =
 - Mathematical assignment 309
 - Parameter delimiter 87
 - URL delimiter 88
 - Variable assignment 250
- ==
 - Boolean equality 276
 - Mathematical equality 310
 - String equality 283
- >
 - Mathematical greater than 310
 - String order 283
- >=
 - Mathematical greater than or equal 310
 - String order 283
- >>
 - String contains 283
- ?
 - Regular expressions 300
 - URL delimiter 88
- ?>
 - LassoScript delimiter 448
- []
 - Regular expressions 299
 - Tag delimiter 87
- \
 - Escape Character 575
 - Line Endings 401, 644
 - Regular Expressions 299
- ^
 - Regular expressions 299
- { }
 - Compound Expressions 87
 - Regular expressions 300
- |
 - Regular expressions 300
- ||
 - Boolean or 275
 - Logical expressions 86

A

- Abbreviation 70
- [Abort] 274
- Absolute Paths 51
- Accessing PDF File Information 566
- Action.Lasso 43
 - HTML forms 115
 - Paths 52
- [Action_Param]
 - Database searches 141
 - Inline actions 113
- [Action_Params]
 - Displaying the current action parameters 121
 - HTML forms 114
 - Inline actions 114
 - Linking to data 163
 - Results schema 123
- Action Errors 410
- Action Methods 42
- Action Parameters 120
- Add 172
 - Requirements 175
- Adding Content to Table Cells 584
- Adding Records 174
 - Classic Lasso 173
 - FileMaker Pro 175
 - Lasso MySQL 175
 - MySQL 175
 - Security 173
 - Using an HTML form 177
 - Using an inline 176
 - Using a URL 177
- [Admin_ChangeUser] 428
- [Admin_CreateUser] 428
- [Admin_GroupAssignUser] 428
- [Admin_GroupListUsers] 428
- [Admin_GroupRemoveUser] 428
- [Admin_ListGroups] 428
- Administration Tags 428
- AND 144
 - Performing an and search 144
- [Array] 340
- [Array->Get] 271
- [Array->Merge]
 - Parameters 347
- [Array->Size] 271
- [Array] 253
- Arrays 77, 339, 340
 - Automatic string casting 281
 - Common arrays 358

- Compressing an array 371
- Converting a string to an array 297
- Creating 340
- Creating an empty array 341
- Creating a pair array 350
- Finding an element 348
- Finding a pair within an array 350
- Getting an element 343
- Getting the size 343
- Inserting an element 344
- Iterating through an array 343
- Joining into a string 346
- Looping through an array 343
- Members tags 342
- Merging arrays 347
- Pair arrays 350
- Passing values into an inline 119
- Removing an element 345
- Setting an element 344
- Sorting 351
- Types 340
- Array Parameters 119
- [Auth] 426
- [Auth_Admin] 426
- Authentication 30
- Authentication Tags 391

B

- Barcodes 590
- Base 64 Encoding 364
- Binary Formats 48
- Binary Operations 314
- Bit Operations 314
- Blowfish 368
 - Seeds 368
 - Storing data securely 369
- [Boolean] 253, 275
 - Data Type 275
 - False 275
 - Symbols 275
 - True 275
- Boolean Operations 314
- Brackets 56
- bw 143
- [Bytes] 253
- Byte Order Mark 40

C

- [Cache] 497
- [Cache_Empty] 501

- [Cache_Fetch] 501
- [Cache_Object] 500
- [Cache_Store] 500
- Caching 496
- [Case] 267
- Casting
 - String 281
- Cellular Phones 511
- CGI 484
- Character Encoding 40, 137, 172
- character set 503
- Character Sets 29
- [Checked] 197, 226
 - Displaying selected values 230
- Check Boxes 201
- Classic Lasso
 - Adding records 173
 - Database searches 138
 - Deleting records 173
 - Tokens 117
 - Updating records 173
- [Client_Address] 508
- [Client_Browser] 508
- [Client_ContentLength] 507
- [Client_ContentType] 507
- [Client_CookieList] 492
- [Client_Cookies] 492
- [Client_FormMethod] 507
- [Client_GETArgs] 507
- [Client_GETParams] 507
- [Client_Headers] 507
- [Client_IP] 508
- [Client_Password] 507
- [Client_POSTArgs] 507
- [Client_POSTParams] 507
- [Client_Type] 508
- [Client_Username] 507
- Client Tags 508
- cn 143
- Color
 - Creating a random color 319
- Command Tags 65
 - Action tags 110
 - Email sending tags 457
- Comments
 - LassoScript 449
- Complex Expressions 83
- Compound Expressions 58
- [Compress] 371
- Compression 371
 - Compressing an array 371
 - Compressing a string 371

- Conditional Expressions 84
 - Symbols 85
- Conditional Logic 263
 - Complex conditionals 266
 - If else conditionals 264
 - Iterations 272
 - Loops 268
 - Nested conditionals 266
 - Select statements 267
 - While loops 273
- Configuration Tags 441
- Container Tags 64
 - LassoScript 449
 - Link tags 161
- [Content_Type] 28, 29, 504
 - Serving images and multimedia 478
 - Serving WML 512
 - XML data 551
- Content Type 504
- Control Tags 425
- [Cookie] 492
- [Cookie_Set] 492
 - Parameters 493
- Cookies 29, 491
 - Checking for cookie support 496
 - Retrieving cookies 494
 - Setting cookies 492
- Creating Barcodes 590
- Creating PDF Documents 559, 564
- Creating Tables 582
- Creating Text Content 567
- Credit Cards
 - Checking whether a number is valid 444
- Custom Errors
 - Using the [Protect] ... [/Protect] tags 423
- Custom Error Pages 411
 - Defining 413
 - Testing 414
- Custom Tags
 - XML-RPC 542, 547

D

- Database 130
- Database 50
 - [Database_ChangeField] 203
 - Parameters 206
 - [Database_CreateField] 203
 - Parameters 206
 - [Database_CreateTable] 203
 - Parameters 204
 - [Database_Names] ... [/Database_Names]
 - Listing available databases 127
 - [Database_RealName] 131
 - [Database_RemoveField] 203
 - [Database_RemoveTable] 203
 - [Database_SchemaNameItem] 240
 - [Database_SchemaNames] 240
 - [Database_TableNames] ... [/Database_TableNames]
 - Listing available tables 128
- Databases
 - Listing available databases 127
 - Listing fields 128
 - Required fields 129
- Database Actions 110
 - Action parameters 120
 - Displaying the current parameters 121
 - Error codes 682
 - FileMaker Pro error codes 690
 - Finding all records 111
 - HTML forms 113
 - Inline method 109
 - JDBC error codes 694
 - Lasso MySQL error codes 686
 - Response tags 115
 - Results 124
 - Searching for records 112
 - Tags 110
 - Tokens 117
- Database Errors 410
- Database Field Paths 53
- Database Schema
 - Showing 125
- Database Searches 136
 - Classic Lasso 138
 - Complex queries 146
 - Detail links 168
 - Displaying data 153
 - Displaying results from a named inline 155
 - Displaying results out of order 155
 - Displaying search results 154
 - Error reporting 137
 - Field operators 142
 - Finding all records 150
 - Finding random records 152
 - HTML forms 141
 - Limiting returned fields 149
 - Linking to data 156
 - Logical operators 144
 - Manipulating the found set 148

- Navigation links 164
- Operators 142
- Performing a logical not search 145
- Random sorting 153
- Results 147
- Returning part of a found set 149
- Returning unique field values 151
- Searching records 139
- Security 138
- Security command tags 139
- Sorting links 167
- Sorting results 147, 148
- Specifying field operators 143
- Specifying username and password 139
- Using a logical and operator 144
- Using a logical or operator 145
- Using inline tags 140
- Data Output 247
- Data Type
 - Boolean 275
- Data Types 253
 - Casting 254, 257
 - Decimal 306
 - Integer 306
 - Returning the type of a variable 438
 - XML 522
 - XML-RPC 539
- [Date] 253, 325
- [Date->Add] 335
- [Date->Day] 330
- [Date->DayofWeek] 330
- [Date->DayofYear] 330
- [Date->Difference] 336
- [Date->GMT] 330
- [Date->Hour] 330
- [Date->Millisecond] 330
- [Date->Minute] 330
- [Date->Month] 330
- [Date->Second] 330
- [Date->Subtract] 335
- [Date->Time] 330
- [Date->Week] 330
- [Date->Year] 330
- [Date_Add] 334
- [Date_Difference] 334
- [Date_Format] 325
- [Date_GetLocalTimeZone] 325
- [Date_GMTToLocal] 325
- [Date_LocalToGMT] 325
- [Date_Maximum] 325
- [Date_Minimum] 325
- [Date_SetFormat] 325
- [Date_Subtract] 334
- Dates 76, 321
 - Accessors 330
 - Formatting 327
 - Math Operations 333, 337
- Date Data Type 321
- Date Format Symbols 327, 329
- Date Math Symbols 337
- Date Math Tags 333, 335
- Date Tags 322
- Daylight Savings Time 322, 333
- [Decimal] 253, 307
- [Decimal->SetFormat] 311
 - Parameters 312
- Decimals 75, 306
 - Assignment symbols 309
 - Automatic string casting 281
 - Casting 307
 - Comparing values 310
 - Comparison symbols 310
 - Formatting 311
 - Formatting as currency 312
 - Member tags 311
 - Random numbers 318
 - Rounding values 317
 - Scientific notation 312
 - Substitution tags 316
 - Trigonometry 319
 - Using assignment symbols 309
 - Using mathematical symbols 308
- [Decode_Base64] 367
- [Decode_URL] 367
- [Decompress] 371
- [Decrypt_BlowFish] 368
- [Define_Tag]
 - XML-RPC 542, 547
- Delete 172
 - Requirements 183
- Deleting Records 183
 - Classic Lasso 173
 - Deleting several records 184
 - Security 173
 - Using an inline tag 183
- Delimiters 87
 - LassoScript 448
- Detail Links 156
 - Inline Lasso 168
- Displaying data 153
- Distinct 147, 192
- Documentation 21
- Documentation Conventions 22

- Document Type Definition 520, 521
- Domain Name Server 443
- Downloading Files 489, 490
- Downloading Graphics Direct to PDF Pages 587
- DTD 521
 - See also Document Type Definition*
- Duplicate 172
 - Requirements 185
- Duplicating Records 185
 - Using an inline tag 186
- [Duration] 332
- [Duration->Day] 332
- [Duration->Hour] 332
- [Duration->Minute] 332
- [Duration->Month] 332
- [Duration->Second] 332
- [Duration->Week] 332
- [Duration->Year] 332
- [Duration] 253
- Durations 77, 321
 - Math Operations 333, 337
- Duration Data Type 321
- Duration Math Tags 335
- Duration Tags 331

E

- [Else] 264, 265
 - Complex conditionals 266
- Email 455
 - Attachments 460
 - Command tags 457
 - Monitoring in Lasso Administration 455
 - Multiple recipients 458
 - Sending a message 457
 - Sending HTML messages 459
- [Email_Send]
 - Parameters 456
- [Encode_Base64] 367
- [Encode_Break] 367
 - HTML encoding 363
- [Encode_HTML] 367
 - HTML encoding 363
- [Encode_Set]
 - Encoding for WML 514
 - Setting encoding within a LassoScript 450
- [Encode_Set] ... [/Encode_Set] 366
 - Setting default encoding 366
- EncodeSmart 365
 - HTML encoding 363
- [Encode_Smart] 367
 - HTML encoding 363
- [Encode_SQL] 367
- [Encode_StrictURL] 367
 - URL encoding 364
- [Encode_URL] 367
 - URL encoding 364
- [Encode_XML] 367
 - XML encoding 364
- EncodeBreak 365
 - HTML encoding 363
- EncodeHTML 365
 - Default encoding 362
 - HTML encoding 363
- EncodeNone 365
- EncodeStrictURL 365
 - URL encoding 364
- EncodeURL 365
 - URL encoding 364
- EncodeXML 365
 - Encoding for WML 514
 - XML encoding 364
- Encoding 361
 - Base 64 encoding 364
 - Controls 366
 - Default encoding 362
 - HTML 363
 - HTML encoding 363
 - Keywords 365
 - LassoScripts 362
 - Rules for encoding 362
 - Substitution tags 362
 - Tags 367
 - URL encoding 364
 - Using encoding tags 367
 - WML 514
 - XML 551
- Encoding Keywords 72
- [Encrypt_BlowFish] 368
- [Encrypt_MD5] 368
- Encryption 368
 - BlowFish 368
 - MD5 hash function 368
 - Storing and checking passwords 370
 - Storing data securely 369
- ENUM MySQL Data Type 196
- EQ 143
- [Error_CurrentError] 415
- [Error_NoRecordsFound] 417
- [Error_SetErrorCode] 415
- [Error_SetErrorMessage] 415

- Errors
 - Types 410
- Error Codes 681
 - FileMaker Pro 690
 - JDBC 694
 - Lasso MySQL 686
- Error Control 385, 409, 418
 - Checking for an error 418
 - Displaying the current error message 415
 - Executing code if an error is encountered 420
 - Fail tags 421
 - Handle tags 419
 - Outputting debugging messages 420
 - Post-processing 420
 - Protecting a portion of a page 422
 - Protect tags 422
 - Reporting an error 421
 - Response tags 414
 - Setting the current error message 416
 - Standard error tags 417
 - Tags 415, 419
- Error Messages 411
 - Built-In 411
 - Custom 412
- Error Pages 414
 - Custom 412
- Error Reporting 244
 - Adding records 172
 - Checking for an error 137
 - Database searches 137
 - Deleting records 172
 - Displaying the current error 137
 - Updating records 172
- [Event_Schedule] 432
 - Parameters 432
 - Scheduled actions 45
- Event Administration 432
- Event Tags 432
- EW 143
- Example PDF Files 592
- Expressions 79
- Extensible Markup Language 521
- Extensible Stylesheet Language 521

F

- False 275
- [Field] 154, 218
 - Database searches 141
 - Displaying results out of order 155
 - Displaying search results 154
 - Returning related fields 219
- [Field_Name] 270
 - Listing fields 128
 - Parameters 128
- Fields
 - Paths 53
 - Required fields 129
- Field Operators 142
- [File->Close] 406
- [File->Delete] 406
- [File->Get] 406
- [File->GetPosition] 406
- [File->IsOpen] 406
- [File->MoveTo] 406
- [File->Name] 406
- [File->Open] 406
- [File->Path] 406
- [File->Read] 406
- [File->SetMode] 406
- [File->SetPosition] 406
- [File->SetSize] 406
- [File->Size] 406
- [File->Write] 406
- [File] 405
- [File_Copy] 396
- [File_Create] 396
- [File_CreationDate] 396
- [File_CurrentError] 396
- [File_Delete] 397
- [File_Exists] 397
- [File_GetLineCount] 397
- [File_GetSize] 397
- [File_IsDirectory] 397
- [File_ListDirectory] 397
- [File_ModDate] 397
- [File_Move] 397
- [File_Read] 397
- [File_ReadLine] 397
- [File_Rename] 397
- [File_SetSize] 397
- [File_Uploads] 403
- [File_Write] 398
- FileMaker Pro 211
 - Adding a record through a portal 221
 - Adding a record with repeating fields 223
 - Adding records 175
 - Checking for databases 215
 - Compatibility tips 214
 - Deleting a record through a portal 222

- Deleting repeating field values 225
 - Displaying a value list 226
 - Displaying data 218
 - Duplicating a record 185
 - Error codes 690
 - Executing a script 235
 - Field operators 142
 - Images 231
 - Key fields 216
 - Listing databases 215
 - Logical operators 144
 - Performance tips 213
 - Portals 220
 - Record IDs 216
 - Referencing a record by ID 216
 - Related fields 219
 - Repeating fields 222
 - Returning a random record 152
 - Returning the current record ID 216
 - Returning values from a repeating field 223
 - Scripts 234
 - Serving an image 231
 - Sorting records 217
 - Terminology 212
 - Updating a record within a portal 221
 - Updating a record with repeating fields 224
 - Value lists 226
 - XML templates 552
 - Files 383
 - Error codes 684
 - Management 48
 - Paths 26, 393
 - Security 395
 - Tags 392
 - File Permissions 463
 - File Suffixes 385, 395
 - File Uploads 402
 - FindAll 136
 - Inline action 111
 - Requirements 151
 - FMScript 234
 - FMScriptPre 234
 - FMScriptPreSort 234
 - [Form_Param]
 - See [Action_Param]*
 - Format Files 37, 244
 - Action methods 42
 - Character Encoding 40
 - Editing 40
 - File management 48
 - Functional types 41
 - HTML form actions 43
 - Inline actions 44
 - Naming 39
 - Output formats 47
 - Post-processing 420
 - Scheduled actions 45
 - Securing 46
 - Specifying paths 50
 - Startup actions 46
 - Storage types 38
 - Unicode 40
 - URL actions 42
 - Forms 261
 - See also HTML Forms*
 - Form Parameters 262
 - [Found_Count]
 - Displaying the current found count 125
 - FT 143, 190
 - FTP 489
 - [FTP_GetFile] 490
 - [FTP_GetListing] 490
 - [FTP_PutFile] 489
 - Full-Text Search 191
- ## G
- GET Method 33
 - GIF
 - Serving image files 478
 - GT 143
 - GTE 143
- ## H
- [Handle] ... [/Handle] 419
 - [Header] ... [/Header] 504
 - Header Tags 503, 504
 - Hexadecimals 314
 - Creating a random color 319
 - Host Name 26
 - Looking up an IP address 443
 - HTML 521
 - Encoding 363
 - Output formats 47
 - Sending email messages 459
 - [HTML_Comment] 247
 - HTML Delimiters 88
 - HTML Format Files 38
 - HTML Forms 32, 33
 - Actions 43

- Adding a record 177
- Creating a pop-up menu 227
- Creating radio buttons 228
- Executing a FileMaker Pro script 235
- Format files 115
- GET method 33
- Inline actions 113
- Input syntax 59
- POST method 33
- Response tags 115
- Searching databases 141
- Setting values 116
- Updating a record 180
- HTTP 488
- [HTTP_GetFile] 488
- HTTPS 484, 486
- HTTP Content and Controls 483
- HTTP Delimiters 88
- HTTP Requests 26
- HTTP Responses 27
- HyperText Markup Language 521

I

- [If] 264, 265
 - Complex conditionals 266
 - Error control 265
 - LassoScript 265
 - Nested conditionals 266
- Illegal Characters 88
- [Image->AddComment] 467
- [Image->Annotate] 473
- [Image->Blur] 471
- [Image->Comments] 466
- [Image->Composite] 474
- [Image->Contrast] 470
- [Image->Crop] 469
- [Image->Depth] 465
- [Image->Describe] 466
- [Image->Enhance] 471
- [Image->File] 466
- [Image->FlipH] 469
- [Image->FlipV] 469
- [Image->Format] 465
- [Image->Height] 465
- [Image->Modulate] 470
- [Image->Pixel] 466
- [Image->ResolutionH] 465
- [Image->ResolutionV] 465
- [Image->Rotate] 469
- [Image->Scale] 469
- [Image->Sharpen] 471
- [Image->Width] 465
- image/gif 478
- image/jpeg 478
- [Image] 462, 464
- [Image_URL] 231
 - Serving an image 231
- ImageMagick 462
- Images 461
 - FileMaker Pro 231
 - Generating the path to a file 477
 - MIME types 478
 - Serving an image file 479
 - Serving an image from FileMaker Pro 231
- Image Formats 463
- [IMG] 231
 - Parameters 233
 - Serving a FileMaker Pro image 233
- [Include] 386
- [Include_Raw]
 - Serving images and multimedia 478
- [Include_Raw] 386
- [Include_URL] 484, 544
 - Parameters 485
- Includes 383
- Include Paths 384
- Include URLs 484
- [Inline] ... [/Inline] 110
 - Actions 44
 - Action parameters 120
 - Adding a record 176
 - Array parameters 119
 - Checking for an error 418
 - Database actions 109
 - Deleting a record 183
 - Deleting several records 184
 - Displaying results from a named inline 155
 - Displaying search results 154
 - Duplicating a record 186
 - Executing a FileMaker Pro script 235
 - FindAll action 111
 - Finding all records 151
 - HTML forms 113
 - Linking to data 163
 - Nesting tags 118
 - Passing array parameters 119
 - Search action 112
 - Searching databases 140
 - Specifying field operators 143
 - Specifying username and password 139

- Updating a record 179
- Updating several records 182
- InlineName
 - Displaying results 155
- Inline Lasso 109
 - Detail links 168
 - Navigation links 164
 - Sorting links 167
- Inline Tag 110
- Installation Problems 410
- Integer
 - Substitution tags 316
- [Integer] 306
- [Integer->BitAnd] 313
- [Integer->BitClear] 313
- [Integer->BitFlip] 313
- [Integer->BitNot] 313
- [Integer->BitOr] 313
- [Integer->BitSet] 313
- [Integer->BitShiftLeft] 313
- [Integer->BitShiftRight] 313
- [Integer->BitTest] 313
- [Integer->BitXOr] 313
- [Integer->SetFormat] 313
 - Parameters 314
- [Integer] 253
- Integers 75, 306
 - Assignment symbols 309
 - Automatic string casting 281
 - Bit operations 314
 - Casting to integer 306
 - Comparing values 310
 - Comparison symbols 310
 - Formatting 313
 - Formatting as hexadecimals 314
 - Hexadecimal output 314
 - Member tags 313
 - Random numbers 318
 - Rounding numbers 317
 - Using assignment symbols 309
 - Using mathematical symbols 308
- IP Address
 - Looking up a host name 444
- ISO-8859-1 28, 40, 137, 172
- ISO 8859-1 606
- [Iterate] ... [/Iterate] 272, 273
 - Array elements 272
 - Iterating through an array 343
 - Iterating through a map 354
 - String characters 273
- iText 558

J

- JavaScript
 - Not processing square brackets 436
- JDBC 237
 - Certification 238
 - Data sources 237
 - Error codes 694
 - Tips for usage 238
- JDBC Schema Tags 239
- JPEG
 - Serving image files 478

K

- KeyField
 - Using with FileMaker Pro 175
 - Using with MySQL 175
- [KeyField_Value]
 - Using with FileMaker Pro 179
 - Using with MySQL 179
- Keywords
 - Encoding 365

L

- [Lasso_DataSourceIsFileMaker] 215
- [Lasso_DatasourceIsFileMaker] 441
- [Lasso_DatasourceIsLassoMySQL] 189
- [Lasso_DatasourceIsLassoMySQL] 441
- [Lasso_DatasourceIsMySQL] 189
- [Lasso_DatasourceIsMySQL] 441
- [Lasso_DatasourceModuleName] 441
- [Lasso_TagExists] 441
- [Lasso_TagModuleName] 441
- [Lasso_UniqueID] 445
- [Lasso_Version] 441
- LassoScripts 57, 447
 - Comments 449
 - Container tags 449
 - Converting from square bracket syntax 450
 - Delimiters 448
 - Encoding 362
 - Returning values 448
 - Setting default encoding 366, 450
 - Single tag 448
 - Suppressing output 450
 - Syntax 448
- Lasso 6 Documentation 21
- Lasso Administration
 - Monitoring email 455
- Lasso MySQL

- Adding records 175
- Error codes 686
- Field operators 142
- Logical operators 144
- Random sorting 153
- Returning unique field values 151
- SQL encoding 364
- Lasso Service 49
 - Paths 53
- Lasso Web Server Connector 48
- Latin-1 29, 40, 137, 172, 606
- LDML 55, 245
 - Converting to LassoScript 450
 - Format files 39
 - Legacy tags 676
 - Syntax types 56
 - Tag categories 67
 - Tag listing 655
 - Tag types 60
- LDML 6 Reference 91
 - Browsing 98
 - Comments 102
 - Components 92
 - Searching 93
 - Sections of the interface 92
 - Tag detail 100
 - Tag listing 104
- LDML 6 Tag Language 55
- LDML Reference
 - Navigation 93
- Leap Years 322
- libCURL 484, 488, 489
- [Library] 386
- Library Files 41, 384
- Line Endings 401
- [Link_CurrentAction] ... [/Link_CurrentAction] 161
 - Linking to the current record 168
- [Link_CurrentActionURL] 160
- [Link_Detail] ... [/Link_Detail] 161
 - Linking to the current record 168
- [Link_DetailURL] 160
- [Link_FirstGroup] ... [/Link_FirstGroup] 161
 - Creating sort links 167
- [Link_FirstGroupURL] 160
- [Link_FirstRecord] ... [/Link_FirstRecord] 161
- [Link_FirstRecordURL] 160
- [Link_LastGroup] ... [/Link_LastGroup] 161
- [Link_LastGroupURL] 160
- [Link_LastRecord] ... [/Link_LastRecord] 161
- [Link_LastRecordURL] 160
- [Link_NextGroup] ... [/Link_NextGroup] 161
 - Creating next links 164
- [Link_NextGroupURL] 160
- [Link_NextRecord] ... [/Link_NextRecord] 161
- [Link_NextRecordURL] 160
- [Link_PrevGroup] ... [/Link_PrevGroup] 161
 - Creating previous links 164
- [Link_PrevGroupURL] 160
- [Link_PrevRecord] ... [/Link_PrevRecord] 161
- [Link_PrevRecordURL] 160
- Linking to Data 156
 - Container tags 161
 - Tag parameters 157
 - URL tags 160
- Linking to PDF Files 597
- List Array 340
- Literals 80
- [Log] 389
- [Log_Critical] 390
- [Log_Detail] 390
- [Log_SetDestination] 391
- [Log_Warning] 390
- Logging 383, 388
 - Changing log destination preferences 392
 - Destination codes 392
 - Logging to file 388
 - Logging to internal database 389
 - Message level codes 391
 - Preferences 391
 - Resetting log destination preferences 392
- Logical Errors 410
- Logical Expressions 85
 - Symbols 86
- Logical Operators 144
 - Performing an and search 144
 - Performing an or search 145
 - Performing a not search 145
- [Loop] ... [/Loop] 268, 270
 - Array elements 271
 - Display field names 270
 - Looping through an array 343
 - Looping through a map 354
 - Parameters 269

[Loop_Abort] 269, 270, 274
 [Loop_Count] 269, 270, 274
 Lower Case
 Strings 290
 LT 143
 LTE 143

M

Mac OS X File Permissions 396
 [Map] 253
 Maps 78, 339, 352
 Automatic string casting 281
 Common maps 358
 Comparison to pair arrays 356
 Creating a map 352
 Displaying an element 356
 Getting a value 353
 Inserting an element 355
 Iterating through a map 354
 Looping through a map 354
 Member tags 353
 Removing an element 355
 Math 305
 See also Decimals
 See also Integers
 Addition 309
 Expressions 82
 Scientific notation 312
 Symbols 83, 308
 Trigonometry 319
 [Math_Abs] 316
 [Math_ACos] 319
 [Math_Add] 316
 [Math_ASin] 319
 [Math_ATan] 319
 [Math_ATan2] 319
 [Math_Ceil] 316
 Rounding numbers 317
 [Math_ConvertEuro] 316
 [Math_Cos] 319
 [Math_Div] 316
 [Math_Exp] 319
 [Math_Floor] 316
 Rounding numbers 317
 [Math_Ln] 319
 [Math_Log10] 319
 [Math_Max] 316
 [Math_Min] 316
 [Math_Mod] 316
 [Math_Mult] 316
 [Math_Pow] 319
 [Math_Random] 316
 Parameters 318
 [Math_RInt] 316
 Rounding numbers 317
 [Math_Roman] 316
 [Math_Round] 316
 Rounding numbers 317
 [Math_Sin] 319
 [Math_Sqrt] 319
 [Math_Sub] 316
 [Math_Tan] 319
 -MaxRecords 130, 147
 MD5 368
 MD5 Hash Function
 Storing and checking passwords 370
 Member Tags 63, 80, 260
 Decimal tags 311
 Integer tags 313
 Member Tag Types 261
 MIME Type
 Image files 478
 Miscellaneous Tags 443
 Multimedia 461
 Generating the path to a file 477
 MIME types 478
 Serving a multimedia file 479
 MySQL
 See Lasso MySQL
 Adding and updating records 195
 Adding records 175
 Creating fields 205
 Creating tables 202
 Data sources 187
 Error codes 686
 Field operators 142
 Field types 207
 Logical operators 144
 Random sorting 153
 Returning unique field values 151
 Searching records 190
 Search command tags 192
 Search field operators 190
 Security 188
 SQL encoding 364
 Tags 189
 Tips for usage 188

N

Named Inlines
 Displaying results 155
 Name Server 443

- Navigation Links 156
 - Inline Lasso 164
- NEQ 143
- Nesting Tags 118
- [NoProcess] ... [/NoProcess] 435
- NOT 144
 - Performing a not search 145
- Nothing 120
- NRX 143, 191
- [NSLookup] 443
- Null
 - Data type 437
 - Value 194, 196
- [Null->DetachReference] 437
- [Null->Dump] 437
- [Null->FreezeType] 437
- [Null->FreezeValue] 437
- [Null->Properties] 437
- [Null->Serialize]
 - Compressing an array 371
- [Null->Serialize] 437
- [Null->Type] 253, 437
- [Null->UnSerialize] 437
- [Null] 437

O

- OpBegin
 - Complex queries 146
- OpEnd
 - Complex queries 146
- OpenSSL 484
- Operating System Errors 410
- Operator 142
- OperatorBegin 142
- OperatorEnd 142
- OperatorLogical 142
 - Performing an and search 144, 145
- Operators
 - Database searches 142
 - Field operators 142
 - Logical operators 144
- Optimizing Tables 209
- [Option] 197, 226
 - Creating a pop-up menu 228
- OR 144
 - Performing an or search 145
- [Output] 247
 - Automatic string casting 281
 - Returning values from a LassoScript 448
- [Output_None] 247

- Suppressing LassoScript output 450
- Outputting Values 247
- Output Formats 47
- Output Suppressing 248

P

- Page Variables 439, 440
- [Pair] 253
- Pairs 339, 356
 - Automatic string casting 281
 - Creating a pair 357
 - Displaying an element 358
 - Getting an element 358
 - Member tags 357
 - Setting an element 358
- Pair Arrays 340, 350
 - Comparison to maps 356
- Parameters
 - Array objects 119
- Password 139
- Paths
 - Absolute 51
 - Action.Lasso 52
 - Database fields 53
 - Lasso Service 53
 - Relative 51
 - Specifying 50
- PDF 557
 - Output Formats 47
- PDF, Introduction to Creating PDF Files 462, 558
 - [PDF_Barcode] 590
 - [PDF_Doc] 464, 559, 564
 - [PDF_Doc->AddChapter] 563
 - [PDF_Doc->AddCheckBox] 576
 - [PDF_Doc->AddComboBox] 576
 - [PDF_Doc->AddHiddenField] 577
 - [PDF_Doc->AddList] 573
 - [PDF_Doc->AddPage] 563
 - [PDF_Doc->AddPasswordField] 576
 - [PDF_Doc->AddRadioButton] 576
 - [PDF_Doc->AddRadioGroup] 576
 - [PDF_Doc->AddResetButton] 577
 - [PDF_Doc->AddSelectList] 577
 - [PDF_Doc->AddSubmitButton] 577
 - [PDF_Doc->AddText] 562, 571
 - [PDF_Doc->AddTextArea] 576
 - [PDF_Doc->AddTextField] 576
 - [PDF_Doc->Circle] 588
 - [PDF_Doc->Close] 567
 - [PDF_Doc->CurveTo] 588

- [PDF_Doc->DrawArc] 588
- [PDF_Doc->DrawText] 573
- [PDF_Doc->GetColor] 566
- [PDF_Doc->GetHeaders] 566
- [PDF_Doc->GetMargins] 566
- [PDF_Doc->GetPageNumber] 563
- [PDF_Doc->GetSize] 566
- [PDF_Doc->InsertPage] 565
- [PDF_Doc->Line] 588
- [PDF_Doc->MoveTo] 588
- [PDF_Doc->Rect] 588
- [PDF_Doc->SetColor] 588
- [PDF_Doc->SetFont] 566
- [PDF_Doc->SetLineWidth] 588
- [PDF_Doc->SetPageNumber] 563
- [PDF_Font] 568
- [PDF_Font->GetColor] 569
- [PDF_Font->GetEncoding] 569
- [PDF_Font->GetFace] 569
- [PDF_Font->GetPSFontName] 569
- [PDF_Font->GetSize] 569
- [PDF_Font->GetFullFontName] 570
- [PDF_Font->GetSupportedEncodings] 569
- [PDF_Font->IsTrueType] 569
- [PDF_Font->SetColor] 569
- [PDF_Font->SetEncoding] 569
- [PDF_Font->SetFace] 569
- [PDF_Font->SetSize] 569
- [PDF_Font->SetUnderline] 569
- [PDF_Image] 586
- [PDF_List->Add] 574
- [PDF_Read->PageCount] 564
- [PDF_Read->PageSize] 564
- [PDF_Serve] 598
- [PDF_Table] 582
- [PDF_Table->GetAbsWidth] 583
- [PDF_Table->GetColumnCount] 583
- [PDF_Table->GetRowCount] 583
- [PDF_Table->Insert] 584
- Performance Tips
 - FileMaker Pro 213
- Personal Digital Assistants 511
- Pop-Up Menu 200
- Portable Document Format 557
- [Portal] ... [/Portal] 218
 - Returning portal values 220
- Portals
 - Adding a record through a portal 221
 - Deleting a record through a portal 222
 - FileMaker Pro 220
 - Updating a record within a portal 221

- Port Number 26
- Post-Lasso 41
- Post-Processing 420
- POST Method 33
- Pre-Lasso 41
- [Process] 435
 - Processing code stored in a field 436
 - Processing code stored in a variable 435
- Process Tags 62, 435
- Programming Fundamentals 243
- [Protect] ... [/Protect] 422
- Protocol 26

R

- Random 136
 - Requirements 152
- Random Numbers 318
- [RecordID_Value] 216
- [Records] ... [/Records] 154
 - Database actions 110
 - Database searches 141
 - Displaying results from a named inline 155
 - Displaying search results 154
- Record ID 216
- [Redirect_URL] 275, 487
- [Referrer] 161
- [Referrer_URL] 160
- RegExp
 - See Regular Expressions*
- Regular Expressions 191, 298
 - Combination symbols 300, 301
 - Finding expressions 302
 - Matching symbols 299
 - Replacement symbols 300
 - Replacing expressions 301
- Relative Paths 51
- Remote Procedure Call 521
- [Repeating] ... [/Repeating] 218
 - Returning values from a repeating field 223
- [Repeating_ValueItem] 218
- Repeating Fields 222
 - Adding a record 223
 - Deleting values 225
 - Returning values 223
 - Updating a record 224
- [Repetition] 270
 - Two column display 271
- Request Tags 506

- [Required_Field]
 - Parameters 129
- Response
 - Action.Lasso 115
- [Response_FilePath] 507
- [Response_LocalPath] 507
- [Response_Path] 507
- [Response_Realm] 507
- ResponseAnyError 414
- Response Tags 115
 - Command tags 116
 - Error control 414
- Results
 - Database searches 147
- ReturnField 147
 - Limiting returned fields 150
- RPC 521
- RX 143, 191

S

- Scheduled Events 45
- Scheduling Events 431
- Schema 520, 521
- [Schema_Name] 240
- Scientific Notation 312
- Scripts
 - Executing a Script 235
 - FileMaker Pro 234
- Search 136
 - Inline Action 112
 - Requirements 140
- [Search_Arguments] ... [/Search_Arguments]
 - Displaying search arguments 124
- Searching Databases
 - See Database Searches*
- Security
 - Adding records 173
 - Command tags 139
 - Database searches 138
 - Deleting records 173
 - Duplicating records 173
 - Error codes 683
 - Violations 410
- [Select] 267
 - Data type 268
- [Selected] 197, 226
 - Displaying selected values 229
- [Server_Port] 509
- [Server_Push] 503
- Server Push 502

- Server Tags 509
- Serving PDF Files 597
- Serving PDF Files to Client Browsers 598
- Session 374
- [Session_AddVariable] 374, 375
- [Session_End] 375
- [Session_ID] 375
- [Session_RemoveVariable] 375
- [Session_Start] 374, 375
 - Parameters 376
- Sessions 373
 - Adding variables 378
 - Deleting 379
 - Example 380
 - Removing variables 378
 - Starting a session 376
 - Tags 375
 - Using cookies 377
 - Using links 377
- SET MySQL Data Type 196
- SGML 521
- Show 270
 - Listing fields 128
 - Listing required fields 129
 - Requirements 126
 - Showing database schema 126
- [Shown_Count]
 - Displaying the current shown count 125
- Simple Object Access Protocol 544
- SkipRecords 130, 147
- [Sleep] 435
- Smart HTML Encoding 363
- SOAP 544
- [Sort_Arguments] ... [/Sort_Arguments]
 - Displaying sort arguments 124
- SortField 147
 - Sorting FileMaker Pro results 217
- Sorting
 - Arrays 351
- Sorting Links 156
 - Inline Lasso 167
- Sorting Records
 - FileMaker Pro 217
- SortOrder 147
- SortRandom 147, 192
- Specifying Paths 50
- SQL
 - Encoding 364
- SQL Server
 - XML templates 552
- SQL Statements 129

- Square Brackets 56
 - Converting to LassoScript 450
- SSL 484
- Standard Generalized Markup Language 521
- Startup Actions 46
- Storage Array 340
- Storage Types 38
- [String] 281
- [String->Split]
 - Creating an array 341
- [String] 253
- [String_FindRegExp]
 - Examples 302
- [String_ReplaceRegExp]
 - Examples 301
- Strings 74, 279
 - Assignment 282
 - Automatic casting 281
 - Casting values to string 281
 - Comparisons 285
 - Concatenation 283
 - Converting case 290
 - Converting to an array 297
 - Deleting a substring 284
 - Expressions 81
 - Extracting part of a string 292
 - Finding regular expressions 302
 - Joining an array 346
 - Length 292
 - Manipulation tags 288
 - Regular expressions 298
 - Repeating a string 284
 - Replacing regular expressions 301
 - Splitting a string into an array 341
 - Symbols 82, 282
- Style Sheets 530
- Sub-Tags 80
- Submitting Form Data to Lasso-Enabled
 - Databases 581
- Substitution Tags 60
 - Encoding 362
- Symbols 79, 258
 - Assignment 259
 - Boolean 275
 - Math 308
 - Strings 282
- Synonym 70
- Syntax 56
- Syntax Errors 410
- System.ListMethods 539
- System.MethodHelp 539

- System.MethodSignature 539
- System.MultiCall 539

T

- [Table_RealName] 131
- Tables
 - Listing available tables 128
 - Listing fields 128
 - Required fields 129
- Tags
 - Categories and naming 67
 - Legacy tags 676
 - Listing 655
 - Naming conventions 68
 - Synonyms and abbreviations 70
- [Tags] 439
- Tag Types 60
- Templates
 - XML 552
- Test.Echo 539
- Text Formats 47
- Text Format Files 39
- Time 321
- Time Zone 322
- Tokens 117
- Trigonometry 319
- True 275

U

- UCS Transformation Format 29
- Unicode 29, 40, 137, 172, 503
- Unique
 - Returning unique field values 151
- Unique ID 445
 - See also Sessions*
- Universal Character Set 29
- Update 172
 - Requirements 178
- Updating Records 178
 - Classic Lasso 173
 - Security 173
 - Updating several records 182
 - Using an HTML form 180
 - Using a URL 181
 - Using inline tags 179
- Upgrading
 - Email command tags 457
- Uploading Files 490
- Upper Case
 - Strings 290

- URLs 26, 261
 - Action.Lasso 116
 - Actions 42
 - Adding a record 177
 - Encoding 364
 - Executing a FileMaker Pro script 235
 - Format files 115
 - Link Tags 160
 - Parameters 32, 262
 - Response tags 115
 - Syntax 59
 - Updating a record 181
- UseLimit 147, 192
- Username 139
- Using Fonts 568
- UTF-8 28, 29, 40, 137, 172, 503

V

- [Valid_CreditCard] 444
- [Valid_Email] 444
- Validation Tags 444
- [Valid_URL] 444
- [Value_List] ... [/Value_List] 197, 226
- [Value_ListItem] 197, 226
 - Displaying selected values 229
- Value Lists 196, 226
 - Creating a pop-up menu 227
 - Creating radio buttons 228
 - Displaying all values 226
 - Displaying selected values 229
- [Var] 250
- [Var_Defined] 250
- [Var_Remove] 250
- [Variable] 250
- [Variable_Defined] 250
- Variables 249
 - Checking 252
 - Creating 250
 - Returning data types 254
 - Returning the type of a variable 438
 - Returning values 251
 - Server-side 373
 - Setting 252
- [Variables] 439

W

- WAP 511
 - Tags 515
- [WAP_IsEnabled]

- Checking to see if current browser is
 - WAP enabled 515
- Web Application Servers 34
- Web Browsers 25
 - Authentication 30
 - Cookies 29
- Web Companion 212
- Web Servers 31
 - Connectors 48
 - Errors 410
- [While] 273, 274
- Wireless Application Protocol 511
 - See also* WAP
- Wireless Devices 511
- Wireless Markup Language 511, 521
 - See also* WML
- WML 511, 521
 - Encoding 514
 - Example 516
 - Formatting 512, 513
 - Forms 513
 - Links 513
 - Output formats 47
 - Serving 512

X

- XML 519, 521
 - See also* XML-RPC
 - Attributes 524
 - Children 524, 528
 - Contents 525
 - Customizing templates 554
 - Data type 522
 - Descendants 530
 - Document type definition 520
 - Encoding 363, 551
 - Extracting tags using an XPath 528
 - Extracting tags using XPath 529
 - Formatting 550
 - Format files 38
 - Member tags 523, 533, 534, 535
 - Output formats 47
 - Parameters 528
 - Root tag 528
 - Schema 520
 - Serving 550
 - Templates 552
 - Transformations 530
 - Wireless Markup Language 511
 - XPath 520, 525
- XML-RPC 520, 521, 537

- Built-in data types 539
- Built-in methods 539
- Calling a remote procedure 537, 544
- Calling multiple methods 538
- Calling remote procedured (low-level) 541
- Custom Tags 542, 547
- Data Type 540
- Listing available methods 538
- Processing incoming requests 541, 546
- Processing tags 543
- [XML_Extract] 526
- [XML_RPCCall] 538
- [XML_Serve] 550
- Serving WML 512
- [XML_Transform] 531
- XPath 520, 521, 525
 - Conditional expressions 529
 - Extracting XML Tags 528, 529
 - Simple expressions 527
- XSL 521
 - Transforming XML data 531
- XSLT 521, 530
- XSL Transformations 521