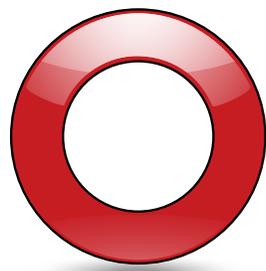


Programmer's Manual



WebDNA
Software Corporation

© 2009, WebDNA Software Corporation

If you have any comments or suggestions about WebDNA, documentation or online help, please send us an e-mail at administration@webdna.us

WebDNA Software Corporation
16192 Coastal Highway
Lewes, DE 19958

www.webdna.us

About This Manual

The *WebDNA Programmer Guide* is designed for web administrators and programmers with a minimum of some knowledge of HTML and web store development.

A second manual, the *WebDNA User Guide*, provides product installation and configuration information in addition to using other features of WebDNA.

Both manuals cover the Macintosh, Windows and UNIX versions of WebDNA. Differences between the versions are noted in the text of the manuals. However, the examples shown in this manual use the Windows notation since the Macintosh and UNIX versions support this convention.

The following assumptions apply to users of the *WebDNA Programmer Guide*:

- Knowledge of the *WebDNA User Guide*
- Familiarity with HTML and your current Web server
- Calling a CGI
- Suffix mapping
- URL encoding

This manual provides a technical introduction to WebDNA, the WebDNA language, and a tutorial on setting up a shopping site from a programming perspective. In addition, it provides a complete reference to the current WebDNA language.

STRUCTURE

The *WebDNA Programmer Guide* provides the following information:

- **Chapter 1 – Understanding WebDNA**
This chapter introduces WebDNA and provides an overview of its structure and usage. Use this chapter to understand the relationship between WebDNA and HTML, and how WebDNA is processed by the web server.
- **Chapter 2 – WebDNA Tutorial**
This chapter is designed specifically to walk you through an example

web site constructed using WebDNA without the aid of our automated site creation templates. Use this chapter to understand how and when WebDNA is used in addition to understanding its tremendous power and flexibility.

- **Chapter 3 – WebDNA Reference**

This chapter provides a printed at-a-glance reference to WebDNA's tags, contexts and commands. Use it to determine which tag, context or command to add for each function to be performed in your environment. There is an HTML equivalent of this with your product install.

- **Chapter 4 – Advanced Uses of WebDNA**

This chapter provides an introduction to more advanced WebDNA topics like how to encrypt templates and securing your WebDNA web site. Use it to better understand how WebDNA can do anything you can imagine a web site doing.

- **Appendices**

This section provides file formats, a glossary, the WebDNA license agreement, and technical support information.

CONVENTIONS

This manual and the *WebDNA User Guide* use the following conventions:

- WebDNA may be used interchangeably as a product line and as a scripting language.
- Since WebDNA is often sensitive about space characters and carriage returns, when giving specific code examples that should not be broken, but do not fit on one line, the continuation on the next line is indented.
- The following icons are used:



Macintosh only



Tips



Windows only



Important Notes



UNIX only

Contents

About This Manual	i
Structure	i
Conventions	ii
Contents	iii
CHAPTER 1 – UNDERSTANDING WEBDNA	1
What Is WebDNA?	1
WebDNA Benefits	2
Definitions	2
Triggers	3
WebDNA Tags	3
WebDNA Contexts	4
WebDNA Commands	7
WebDNA Parameters	8
Related Topics	10
HTML Forms	11
Site Design	12
Database Design	13
Template Design	14
Shopping Cart Transaction Processing	15
Order Collection	17
Order Processing	18
Order Management	19
Using a Text Editor vs. HTML Editor	19
Working with Tables	20
Using Contexts vs. Commands	21
Searching	21
Replacing Commands with Contexts	23
Logging Information	24
CHAPTER 2 – WEBDNA TUTORIAL	26
WebDNA Theory of Operation	26
What is WebDNA	26
How WebDNA Acts on a Web Server Request	27
Preparing Your Site for use with WebDNA	28
Request Processing with WebDNA	29
Development of the TeaRoom Database	30
Tutorial: Overview	30
The TeaRoom Database	31
Step 1: Entering the Site	31
Step 2: Shopping for Products by Category	34

Step 3: Adding Items to the Shopping Cart.....	38
Step 4: Using the Shopping Cart Page	43
Step 5: Using the Product Detail Page	47
Step 6: Using the Purchase/Invoice Page	51
Step 7: Acknowledging the Order	59
WebDNA Lab	60
Where to Go from Here?.....	61
CHAPTER 3 – WEBDNA REFERENCE	62
WebDNA 5.0 At-A-Glance Reference	62
Searching.....	63
[FoundItems] Context.....	63
[LookUp] Tag.....	63
[Search] Context	64
Search Command	65
Searching Comparisons.....	66
[ShowNext] Context	69
[SQL] Context.....	71
Databases	73
[CloseDatabase] Tag	73
[CommitDatabase] Tag	73
[Append] Context	73
[AddFields] Context.....	75
Append Command	77
Delete Command	78
[AppendFile] Context.....	79
[ExclusiveLock] Context	81
[Delete] Tag.....	81
[FlushDatabases] Tag	82
[ListDatabases] Context.....	82
[ListFields] Context.....	83
[LookUp] Tag.....	83
[Table] Context	84
[Quit] Command	89
[Replace] Context.....	90
Replace Command	93
[ReplaceFoundItems] Context	94
[SQL] Context.....	96
Shopping.....	98
Add Command	98
[AddLineItem] Context.....	101
[Cart] Tag	104
Clear Command	104
[ClearLineItems] Tag.....	105

[LinItems] Context	105
NewCartSearch Command	106
NewCart Command.....	107
[OrderFile] Context.....	107
[Purchase] Tag.....	110
Purchase Command	110
[RemoveLinItem] Tag.....	117
Remove Command	117
[SetHeader] Context.....	118
[SetLinItem] Context.....	120
ShowCart Command.....	122
[ValidCard] Tag	123
Showing and Hiding.....	124
[HideIf] Context	124
[HTML1] Context	125
[HTML2] Context	125
[HTML3] Context	126
[If][Then][Else] Context.....	126
[ShowIf] Context.....	128
[Switch][Case] Context.....	128
ShowPage Command	129
Dates and Times.....	131
[Date] Tag.....	131
[Format] Context	132
[Math] Context.....	134
[Time] Tag	140
Text Manipulation.....	140
[BoldWords] Context	140
[Capitalize] Context	141
[ConvertChars] Context.....	142
[CountChars] Context.....	143
[CountWords] Context.....	143
[ConvertWords] Context.....	144
[Decrypt] Context	146
[Encrypt] Context.....	146
[Format] Context	148
[GetChars] Context	149
[Grep] Context.....	152
[Input] Context.....	153
[ListPath] Context.....	154
[ListWords] Context.....	156
[LowerCase] Context.....	157
[Middle] Context	158

[Raw] Context.....	158
Raw Command	159
[RemoveHTML] Context.....	160
[Text] Context.....	160
[UnURL] Context	162
[URL] Context.....	163
[Uppercase] Context	163
Passwords	164
[Authenticate] Tag	164
[Password] Tag	165
[Protect] Tag.....	165
[Username] Tag	165
Files and Folders.....	165
[AppendFile] Context.....	165
[CalcFileCRC32] Tag	166
[CopyFile] Tag.....	167
[CopyFolder] Tag	167
[CreateFolder] Tag	167
[DeleteFile] Tag	167
[DeleteFolder] Tag	168
[FileCompare] Tag	168
[FileInfo] Context	170
[ListFiles] Context.....	171
[MoveFile] Tag	172
[RenameFile] Tag.....	172
[WaitForFile] Context	173
[WriteFile] Context.....	173
Technical.....	174
 [AppleScript] Context.....	174
[Command] Tag	175
 [DDEConnect] Context	175
 [DDESend] Context	176
 [DOS] Context.....	177
 [Shell] Context.....	177
[ElapsedTime] Tag	178
FlushCache Command	178
[FlushDatabases] Tag	179
FlushDatabases Command.....	180
[Interpret] Context	181
[Object] Context	182
Raw Command	184
[Redirect] Tag.....	184

[ReturnRaw] Context.....	185
[Spawn] Context.....	185
[TCPConnect] Context.....	186
[TCPSend] Context.....	187
[Version] Tag.....	189
Browser Info	189
[BrowserName] Tag.....	189
[GetCookie] Tag.....	189
[GetMIMEHeader] Tag.....	190
[IPAddress] Tag.....	190
[IsSecureClient] Tag.....	190
[ListCookies] Context.....	190
[ListMIMEHeaders] Context.....	192
[Referrer] Tag.....	193
[SetCookie] Tag.....	193
[SetMimeHeader] Tag.....	194
XML.....	194
[XMLParse] Context.....	194
[XMLNodes] Context.....	199
[XMLNodesAttributes] Context.....	201
[XSL] Context.....	204
[XSLT] Context.....	206
Miscellaneous.....	217
[ArraySet] Context.....	217
[ArrayGet] Context.....	218
[FormVariables] Context.....	220
[FreeMemory] Tag.....	221
[Function] Context.....	222
[Include] Tag.....	223
[LastRandom] Tag.....	224
[ListVariables] Context.....	224
[Loop] Context.....	226
[Platform] Tag.....	227
[Random] Tag.....	227
[Return] Context.....	228
[Scope] Context.....	231
[SendMail] Context.....	238
Header Fields.....	240
[ThisUrl] Tag.....	241
[Version] Tag.....	241
[!] Comment Context.....	241
Using WebDNA Tags	242
Preferences.....	242

Parameters.....	242
Italic Text.....	243
Paths.....	243
Form Variables.....	244
Using WebDNA Contexts	244
Tag Parameters	245
Context Parameters	245
Context Variables.....	245
Using WebDNA Commands	246
Command= Notation	247
CHAPTER 4 – ADVANCED USES OF WEBDNA	249
Encrypting Templates.....	249
How to Encrypt Templates.....	249
Encrypting the Header Tag	250
Talk List Subscription and Archive.....	251
Using Shared POP Mailbox	251
Generating Online Banner Ads.....	252
Dreamweaver Integration	253
XML Syntax Explanation	253
Security	255
Macintosh WebDNA Security Notes	256
Areas to Watch for security threats.....	262
Uploading Files	262
WebDNA Content Management System	263
What is WebDNA CMS?	263
Using WebDNA CMS.....	264
Advanced WebMerchant Topics.....	266
Using Account Authorizer.....	267
Use of external accounting software with order files	267
APPENDICES	268
File Formats.....	268
Database Format	268
Shopping Cart/Order File Format.....	270
Browser Info.txt Format.....	271
Email Format.....	272
Formulas	273
ISP Sandbox	277
Pre and Post Parse Scripts	281
Triggers.....	283
License and Limited Warranty Agreement.....	286
Support.....	288
Glossary.....	289

Chapter 1 – Understanding WebDNA

What Is WebDNA?

WebDNA is a scripting language for creating web sites. It adds functionality to your web server. WebDNA is used to tell WebDNA products what to do. It exists in HTML files on your server or within URL's sent by your browser.

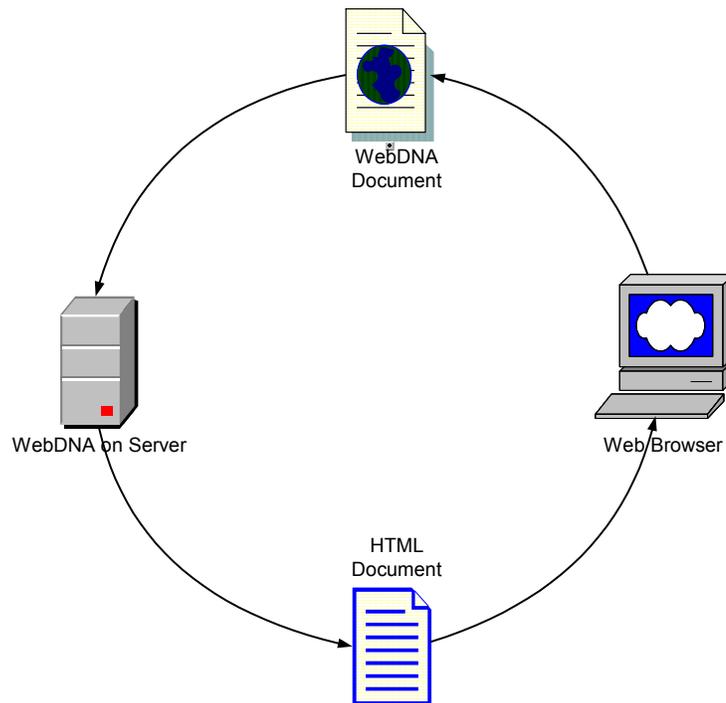


Figure 1. WebDNA Overview

Unlike most Web sites where HTML pages are designed and accessed directly, Web pages accessed from a WebDNA-driven site are processed, otherwise known as interpreted, through WebDNA on a server before being sent to the browser. WebDNA does not require any special client-side applications or web browsers with special capabilities.

WEBDNA BENEFITS

WebDNA adds incredible power to any Web site! WebDNA's syntax and functionality lie between HTML and SQL. At the HTML end, WebDNA shows its ease of use with its simple text file markup tags. At the SQL end, WebDNA can be used to construct sophisticated database searches that result in formatted output. **Note:** WebDNA does not contain a SQL server; it contains its own native powerful database features, plus the ability to connect to other SQL servers through an ODBC interface.

Additional WebDNA benefits include:

- Enhancement to existing web sites by readily integrating other HTML-compatible languages.
- Requires 50% or less coding than the other leading languages, making it faster to code and easy to maintain.
- Utilization of easy-to-learn programming concepts similar to Visual Basic and C/C++.

By enhancing the best aspects of multiple technologies, WebDNA gives Webmasters greater freedom to design dynamic and automated Web sites.

Definitions

The following WebDNA terminology is used in this manual and other WebDNA documentation:

- Triggers
- Tags
- Contexts
- Commands
- Parameters



For a description of the specific WebDNA tags mentioned below, refer to *Chapter 3 – WebDNA Reference*.

TRIGGERS

Definition: *In WebDNA, a trigger provides a mechanism for doing something on a regular timed basis, or when a certain action occurs. Currently, only time-based triggers are available, but in the future new types of triggers will be added to perform an action whenever a database is modified or a template displayed.*

Triggers do their work by simulating a browser hit to a URL. They act in the same way as one who manually uses a browser to reload a page at a particular time each day. Triggers provide the flexibility to create as much complex WebDNA as needed. Once created, the WebDNA can be tested through a browser that visits the subject URL. When a new template is tested and verified, its URL is entered in to a trigger for a predetermined time interval for automatic execution thereafter.



Unless otherwise specified, all WebDNA names are **not** case-sensitive, just like HTML tags are not case-sensitive. By convention, lowercase characters are often used, as they are easier to type.

WEBDNA TAGS

Definition: *A WebDNA tag is just like an HTML tag with one major exception: It never “exists” as far as a browser is concerned. Instead, it is replaced by text (any valid text) on the server by WebDNA before being sent to the browser. Think of WebDNA tags as server tags, and HTML tags as browser (client-side) tags. WebDNA tags are actually used to ‘write’ HTML tags and any other text you need.*

Like HTML tags that exist on their own, WebDNA tags are replaced with some value. However, unlike HTML tags, the value used to replace a WebDNA tag is dynamic and can change depending upon the situation. This dynamic replacement makes your Web pages much more active and interesting. It also means less time will be spent updating pages manually.

An example of a WebDNA tag is the current date on a page. In HTML, the following is inserted:

Today's date is 1/1/00.

However, in order to be accurate, the page would need to be edited every day to insert the correct date; this amounts to a waste of time. What is really needed is a way to insert the date once and have it update itself dynamically, using the following WebDNA tag:

Today's date is [date].

WebDNA tags are not enclosed by angle brackets (< >). Instead, WebDNA tags are enclosed with square brackets ([]). Tags were designed this way intentionally to avoid possible confusion between HTML tags and to make code easier to read. In the above example, when a browser requests the page, the [date] tag is replaced by WebDNA *on-the-fly* with the correct date value. Thus, if today is January 1, 2000, the browser would display the following text:

Today's date is 1/1/00.

Plus, if you had HTML formatting to bold today wrapped around the word "date", the template could look like the following:

Today's date is [date] .

All of your WebDNA tags are replaced by WebDNA with their proper values, and any HTML tags are left intact before being returned to the browser:

Today's date is 1/1/00.

Finally, the browser displays the text as:

Today's date is **1/1/00**



See *Using WebDNA Tags* in Chapter 3 for further information.

WEBDNA CONTEXTS

Definition: A *WebDNA context* encloses a block of text and requires a beginning and ending tag. Like HTML enclosing tags, the ending WebDNA context tag is specified with the name of the context preceded by a forward slash "/". Like WebDNA tags, contexts are enclosed in square brackets rather than angle brackets as well. In a programming sense, the context implies a scope.

The following displays an HTML sample with a WebDNA context example:

```
<h1>Daily Information</h1>
  [showif [date]=01/01/2000]
  Today is the first day of 2000
  [/showif]
```

The [Showif] context hides or shows portions of text based upon a comparison included in the beginning context tag. Because [Showif] is executed on the server, if the equality is not true, then the text is not sent to the browser at all.

However, you can include WebDNA tags within context tags. When this occurs, the innermost tag is substituted before the outer tags and contexts are evaluated. Thus rather than strictly comparing the raw text “[Date]” to “01/01/2000” (which is always false), the value for the WebDNA tag [date] is inserted before the comparison is evaluated. If the date equals “01/01/2000,” then the expression is true and all the text between the beginning and ending [Showif] tags is sent to the browser. The context tags themselves are, of course, collapse and are replaced to the text sent.

Thus the output sent to the browser on 01/01/2000 would be:

```
<h1>Daily Information</h1>
  Today is the first day of 2000
```

As said earlier, WebDNA allows nested contexts by enclosing a context within a context. This lets you create very complex behaviors with combinations of WebDNA tags. Another WebDNA context is the [Raw] context that turns off WebDNA’s interpretation of tags. This is useful if you are writing a description of a WebDNA tag and you want the raw text displayed (and not interpreted).

For example, if you wanted to display the previous example on a page without the [Showif] tag being evaluated, you could enclose it in a [Raw] context. The HTML would appear as:

```
<h1>Daily Information</h1>
  [raw][showif [date]=01/01/2000]
  Today is the first day of 2000
  [/showif]/raw]
```

The [Raw] context surrounding the [Showif] context changes the behavior of [Showif]. Essentially, the [Showif] is not executed because the enclosing [Raw] context turns off the WebDNA interpretation until the ending [Raw] tag. The text sent to the browser would look like:

```
<h1>Daily Information</h1>
  [showif [date]=01/01/2000]
```

Today is the first day of 2000
[/showif]

The [Raw] tags are interpreted and removed, and all the text in between is sent untouched. This sort of context sensitivity is important both when designing your site and when debugging your site if unexpected results occur.



See *Using WebDNA Contexts* in Chapter 3 for further information.

WEBDNA COMMANDS

Definition: *WebDNA commands direct WebDNA to perform various functions. They are sent directly to the application via URL's. WebDNA commands are embedded in a URL and give an explicit context to a particular template being displayed. The context is considered "wrapped" around the entire template.*

WebDNA tags and contexts are contained within HTML template files and are evaluated before being sent back to a browser. When WebDNA is evaluated within a template file, the URL leading to that template looks just like a URL to a specific page. This means all the actions you want to perform must be encapsulated within the template itself.

However, sometimes it is desirable to state what you want to do before evaluating a template. This is done through the use of WebDNA commands. WebDNA commands are embedded in a URL and give an explicit context to a particular template being displayed.

An example is the search command. Since commands are specified in the URL, they are not enclosed in square brackets. Commands are specified as a parameter named "command" after the question mark.

The template being displayed is always described in the path before the question mark. Thus a URL with the search command would look like:

<http://www.server.com/entry.tpl?command=search&searchcriteria>

When WebDNA sees this URL, it uses the "entry.tpl" template to display the result of the search. If the command requires any parameters, they are specified as name and value pairs after an ampersand (&) following the command name and between each pair.

For the most part, there is a corresponding context for every command, though the reverse is not always true since contexts are used in much more complex situations. Also, commands are sometimes specified with slightly different names and parameters due to their inherent differences.

When there is both a command and context that performs a specific function ("search," for example), the decision on when to use a command and when to use a context is up to you. Because changing a URL in a browser is so fast, often programmers use URL-based commands while debugging complex search criteria, then once the desired results are found, they convert

the command to an embedded search context. This issue is discussed later in *Using Contexts Versus Commands*.



See *Using WebDNA Commands* in Chapter 3 for further information.

WEBDNA PARAMETERS

Definition: *A parameter modifies or defines a WebDNA tag, context or command. For example, the [include] tag used to insert the contents of a file at a specified location, needs to know where to find the file to include. In this case, a parameter is used to specify the file location.*

WebDNA uses two methods to specify parameters:

- Method 1: Tag or context requires a single parameter
- Method 2: Tag includes multiple parameters

Method 1: Tag or context requires a single parameter

This method includes tags or contexts requiring a single parameter. The [showif] context falls into this category since it requires a single comparison that must be evaluated. When a single parameter is required, that parameter is placed in the beginning tag after the name of the tag. WebDNA is not case-sensitive, but it is sensitive to extra “blank space” characters. This is quite different from HTML, which often ignores extra space characters. After the WebDNA tag name, a single space character is inserted before the beginning of the parameter. Everything after the single space character is considered part of the parameter. This sensitivity is necessary in case you want to include spaces within a parameter itself.

When a single parameter is required, normally you just list the value for the parameter after the space character. In the [showif] example, the parameter is a comparison. Most of the time, however, the parameter is a path to a file or database. An example of this is the [include] tag for including the contents of external files within a template.

Review the following sample [include] tag:

```
[include header.html]
```

The parameter required for the [include] tag is the file that must be included. In this case, the name of the file is “header.html”. All files and databases are specified just like URL paths. That is, you can specify a relative path

beginning at the template being evaluated, or a full path from the root web server folder. In order to specify a relative path to a file within folder, you would insert the following:

```
[include myfiles/header.html]
```

A folder called “myfiles” must be in the same folder as the template containing this WebDNA. Within that folder, a file named “header.html” is inserted in place of the current WebDNA tag. If you had a global include files folder in the root directory of your web server, you might specify the path as shown below:

```
[include /myfiles/header.html]
```

On Macintosh and UNIX web servers, the preceding forward slash “/” means that the path to the file begins from the root web server folder level no matter where the current template exists. On Windows web servers, the preceding forward slash “/” means that the path to the file begins in the same folder where the program DBServer.exe exists. In both cases, however, the preceding slash lets you specify global, as opposed to relative, locations.

Method 2: Tag includes multiple parameters

On the other hand, the [include] tag can sometimes include multiple parameters. The only required tag is the path to the file. However, you can also include an optional parameter to specify whether to evaluate the included text for WebDNA. In doing this, you tell WebDNA whether to evaluate the included file with a simple “T” or “F” (for true or false). Obviously, you can’t just do the following:

```
[include header.html]
```

WebDNA doesn’t know where the end of the template ends and the true or false begins. When you specify multiple parameters to a tag, context or command, you must specify them in name/value pairs. That is, you must specify the name of the parameter, followed by an equal sign “=” then provide the value for that parameter. Multiple name/value pairs are then strung together with the ampersand character “&” between them. If you have done work with HTML forms, you may recognize this method of specifying parameters. It is identical to the HTML method of specifying parameters contained in a URL. This method was chosen because of its similarity to HTML.

In the above example, the valid method for specifying the tag is:

```
[include file=header.html&raw=F]
```

The two parameter names are “file” and “raw”. These stay the same, but their associated values vary. This allows you to pass as many parameters to the context as you would like without creating confusion. In fact, the [include] tag can often have more than two parameters. Note that this style of naming each parameter (Method 2) is always preferable and reduces confusion compared to unnamed parameters (Method 1).

Another example of this is the search command. Because the multiple parameter specification is similar to that of HTML, the actual URL used to perform a search might be shown as:

```
http://www.server.com/entry.tpl?command=search&db=prod.db  
&eqskudata=1002&eqdescriptiondata=software
```

(Note the line break in the above example is only for readability on this printed page. In a real URL you would never break the line in the middle).

Some characters are not valid in a URL. The same is true for parameters passed to a tag or context. For example, the space character is not valid in a URL. This character must be encoded before being used. This is done primarily to ensure that the URL can be passed through all existing browsers that may only recognize a limited character set. It is also important, however, to reduce confusion. You can't include the ampersand character “&” in the value of a parameter since it would cause WebDNA to think a new set of name/value pairs is coming up. Thus the ampersand character must be encoded if it's in the text for the value of a parameter.

Review *Using Contexts vs. Commands* for a full description of when and how to encode parameters and other strings.

Related Topics

A variety of topics related to creating a WebDNA driven site need to be described. These include:

- HTML Forms
- Site Design
- Database Design
- Template Design

HTML FORMS

Before creating a site with WebDNA, you need to know how HTML forms work and the difference between the POST and GET methods. HTML forms are the standard means by which a visitor's browser sends information to the server. All the data contained in a form, both hidden data and data entered by the visitor, is collected, bundled, and sent to WebDNA via the web server. An HTML form is simply a collection of `<input>` fields with an associated name/value pair. The form's data is sent to the server in a manner similar to the name/value pairs used to specify WebDNA parameters.

In fact, when submitting a form with a method of GET, the data is submitted to the server as a URL as if a big hypertext link was created with all the name/value pairs contained in it.

A simple form can be shown as:

```
<form method="get" action="entry.tpl">
  <input name="variable" value="">
  <input type="submit" value="submit">
</form>
```

The action field of a form should contain a template with the proper Suffix Mapping (the standard is for all filenames ending in ".tpl" to be sent to WebDNA for processing). The text entered in the input field is sent as the value of the field named "variable." Because forms using the GET method are actually sent through the URL, the value of the variable field is encoded automatically by the browser before being sent.

Thus if you typed "a filename" in the input field, the browser would create and link to the following URL:

```
...entry.tpl?variable=a%20filename
```

The browser inserts the question mark character "?" after the action name and before the name/value pairs. Note that the browser encodes the values of the parameters automatically when using the GET method. When there was no WebDNA command explicitly specified, the showpage command is assumed.

If the form was meant to perform a search, then the form and associated URL would appear as:

```
<form method="get" action="entry.tpl">
  <input type="hidden" name="command" value="search">
  <input name="eqskudata" value="">
  <input type="submit" value="submit">
</form>
```

What the browser URL looks

like:...entry.tpl?command=search&eqskudata=a%20filename

WebDNA makes all of the variables passed to a template via a form (in the example above, the variable named "eqskudata") accessible through WebDNA tags. You can include the value of the eqskudata variable in the entry.tpl template using the name of the variable surrounded by square brackets. Thus you could create a [showif] context like the following:

```
[showif [eqskudata]=search criteria]...[/showif]
```

Like hypertext links, however, forms using a GET method are restricted to 256 characters of data. In other words, after the URL is created with all the name/value pairs, the entire URL must be less than 256 characters. This is acceptable for many situations, but if there is any possibility that the link will be more than 256 characters you must use a POST method. Another advantage of using a form method of POST is that the data in the form is not visible in the URL. This can be especially useful if you use a lot of hidden fields in the form (even fields hidden in a form are visible in the URL with the GET method).

The following is a sample form using the POST method and the resulting URL:

```
<form method="post" action="entry.tpl">  
<input name="variable" value="">  
<input type="submit" value="submit">  
</form>
```

What the browser URL looks like:...entry.tpl

Notice that the parameters normally found after the question mark (the <input> parameters) are not visible. With the POST method, the parameters are sent outside the URL so they can contain as much data as necessary. As far as site design is concerned, there isn't any difference between working with form method POST and GET (except the 256 character limit).

SITE DESIGN

The most difficult part of any webmaster's job is overall site design. The following sections are not rules that must be followed. Instead, they are useful suggestions.

Step 1: Outline Your Web Site

In general, it is good to outline your web site in very broad terms to determine its scope. This shouldn't be the name and function of every page; instead, outline 4-5 main areas of your site. While it's good to plan for future capabilities, don't go too far in the future with your expectations. The Web is a rapidly changing environment and sometimes waiting to dive in can lead to unnecessary delays.

Step 2: Define the Steps Needed to Create the Site

Once you have a broad idea for what you want to do, establish the steps necessary to create the site. If you have 4-5 main areas to your site, you will probably need 4 to 5 steps to go through in order to create your site. Each step should implement the basic set of functionality you need in each area. Many times it is easier to take a simple framework and add to it than create a complex framework the first time out.

Step 3: Begin Simply

Especially with sites driven by WebDNA, you should start simply, get the site working, and then gradually make it more complex. This not only reduces the amount of time necessary to get a version of your site up and running, but it allows time for feedback to direct some of the development.

DATABASE DESIGN

A fundamental question with a WebDNA site is database design. Databases play a central role in WebDNA and can be tremendously useful. If designed properly, your site can be totally database driven and updated and modified easily. However, if the database design is particularly poor, your site can be very difficult to adapt to new demands of your site.

It is good to begin with a very simple database and gradually make it more complex. Sometimes it is difficult to know when to use a database and when to hard-code information. For example, suppose your home page has a header at the top that lists your major site categories. This is likely implemented as an [Include] file since it is usually used without modification on many pages. Most sites have a limited number of categories that do not change very often. In this case, it probably isn't useful to make a "site category" database to create the header.

However, you may have sub-headings that can grow continuously within a particular category. For example, the “Product” category may start out with 2 products, then expand to 4 or 10 after 6 months. Initially, the temptation will be to “hard-code” the products page with your existing products. As a first implementation, this is fine. If you expect to add products frequently, though, you might want to create a separate product database. The contents of the product page could then be built from the database automatically. This advantage lets you add new products and have your HTML pages updated automatically.

Suppose you want to create a product database. Rather than try to anticipate all the data you may want to store about the product, start out with a simple database with one field: “name.” Create the product page displaying the names of all the products in that database. It may not be the desired result, but once it works, you will have overcome the major hurdles.

Next, add a field named “description” to the product database. Updating the existing templates to display the new field should be very quick and easy. Continue in this manner until you’re happy with the functionality.

Do not try to over-design your site from the start. This not only delays the time it takes until you see it working (and seeing it work, even on a limited basis, is the best motivation for finishing), but it can often turn out wrong and waste a lot of time.

Any time you find yourself using the same data more than 2-3 times, consider putting that information in a database. Pop-up menus, for example, are often an area where people duplicate code repeatedly.

TEMPLATE DESIGN

Perhaps equal in importance to database design is template design. Template design is important because it is in templates that the bulk of your WebDNA code appears. As with databases, it is best to start simply and gradually improve your templates until they work the way you want them to. Because WebDNA (and HTML) are interpreted languages (evaluated on the fly when the page is requested), they can sometimes be difficult to debug. If the page doesn’t appear in the browser with the information you expect, it can be difficult to track backwards and find the problem.

The easiest way to debug a template is to break it up into “snippets” of code a few lines long and evaluate those smaller files to see if each returns the results you expect. You can gradually “build” your page from these snippets

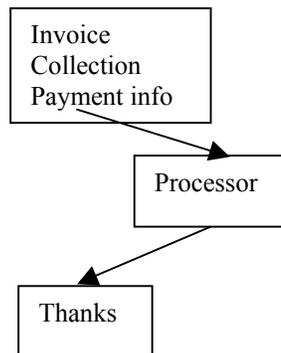
until your results are not as expected and then look at the last snippet you added for problems. One of the difficulties in doing this is realizing that a context can enclose a large amount of text. Because a context requires the ending tag, it may be best to create the code snippets that include an outer context with most of its interior deleted.

Gradually work your way downward or inward in order to complete the template.

Shopping Cart Transaction Processing

Shopping Carts really have two stages they need to move through, Order Collection and Order Processing. In general as a merchant you also need to have tools to manage all of your orders, reporting and details. Order collection and order processing are often tied very tightly together, the closer this relationship the more chance that the merchant can be left high and dry without order collect. The first rule in sales is to get the order and then let the processing happen after the client commits to the purchase. In a web environment, the closest transaction processing style to this is what we call near real time processing. You want to give the user feedback on the processing of their order, but you don't want to get in the way of collecting the order.

Figure 2. Real Time Transaction Processing



In contrast, real time processing is a linear sequence of actions, as shown in Figure 2. Once all the information for an order is collected from the customer (except for payment details), an invoice is generated and the user is asked for payment information. All order information and the payment information is

directed to the merchant's bank while the merchant is off to the side waiting for the customer to return.

On approval of the order, the customer is redirected back to the merchant with the approved notification, a thank you page is displayed to the customer, and the merchant stores the new information that they have received about this order. Under this type of processing style, the merchant is clearly not in control of the transaction, especially if there is an approval problem, where the merchant bank and the customer would continue an exchange while the merchant continues to wait for the result.

Real time processing can be thought of in this way: imagine going to the grocery store and picking up some items to purchase in a shopping cart. Now, you bring the cart to the checkout counter. In order to buy the items, you must talk to a representative of the bank to receive a slip of paper that states you are approved to buy these grocery items. You then bring the slip of paper back to the checker that has your cart of items but they have now gone out on a break or gone home sick or you get lost on the way back to the checker and the merchant loses the sale.

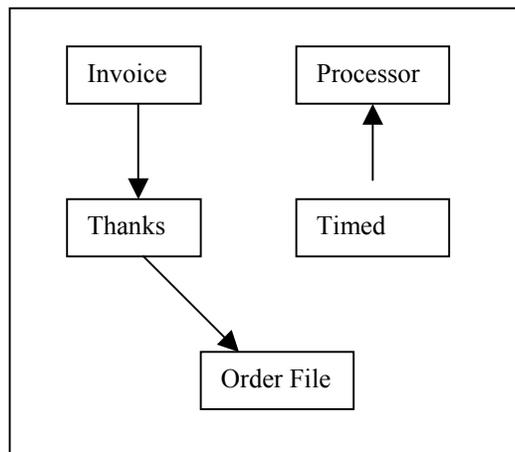


Figure 3. Near Real Time Processing

To avoid these pitfalls, the recommended technique of near real time processing, shown in Figure 3. separates the issue of collecting the order and processing the order. In this type of processing, an order is collected and

acknowledged by the merchant, then, an independent action of processing and authorizing the order can occur. The customer typically is given a page that automatically refreshes at regular intervals. That page (or template) is written to indicate the status of the order while being processed, which results in the indication of a good order (and a link to more details) or a bad order (and a link to help to resolve the problems). In near real time processing, the merchant is in control of all the information about the pending sale and the customer does not have to wait around for the details of processing but is provided with information about their order.

During order processing, merchant bank authorization is initiated by a trigger (a WebDNA timed event) to receive the approval information from the merchant bank. When this information is received, either positive or negative about the order, it is stored back in the order file. WebMerchant further indicates the status of the order by moving it to one of the following directory folders upon receipt of the approval information from the merchant bank: pending, completed, or problems

In near real time processing, you have behavior similar to that in everyday life. In the grocery store example, the checker collects all the information about your purchase and they do the running around to get the bank to approve the sale, staying in control of the transaction.

ORDER COLLECTION

When the customer begins using the shopping cart, desired items are selected and added to the cart. These carts are the beginning of order files and until the shopper commits to the order, they are stored in the ShoppingCarts folder. When the customer has finished collecting all the desired items, he or she clicks the Submit Order (or equivalent) button. The order is collected and processed through the use of WebDNA commands and stored in the order file. The stage at which the order is in is further indicated by what directory the order file is in. When the user submits the payment information, typically the purchase command is issued in WebDNA that moves the order to the Orders directory. This is where the order processing timed event will look for the orders.

The order information is stored in an order file. It is a text file that contains two key pieces of information; all contact information to fulfill the order (such as account information, payment method and contact information) and the list of ordered items. The contact information is at top of the order file and is considered the header portion of the file. This is immediately followed by the list of ordered items, considered the body of the order file. You can find more details on this in the File Formats, Order File section.

Further, a good order processing environment needs to have a unique order number for each order. WebDNA takes that approach one step further, the order number is also the name of the order file. The order number is generated through the use of our 'magic' cart form variable, [cart]. If this form variable is not indicated, WebDNA gives it a unique number.

ORDER PROCESSING

Once the order has been completely collected and the user has received their Thank You page acknowledging the order, the next trigger for order processing will see the order in the Orders directory. Basically the processing is just a timed event that looks at the Orders directory and attempts to get authorization for each order sitting there, stores the result of the authorization into the order file and moves the order file to the appropriate directory, CompletedOrders or Problems. That trigger is calling the template, Dopurchases.tpl in the WebMerchant directory of the store. For a more complete discussion on triggers, see page 3.

Additionally there are intentional 'hooks' (places where you can insert custom instructions) for other processing to occur in this process. There is a file GoodPath.inc that is called when a positive authorization is received for an order. You can place any WebDNA in this file that you want to happen when a good order is received, such as adding to an import log for your accounting package. An example of this is discussed in the Advanced Topics section later in this manual. A similar hook is provided for the BadPath.inc. Remember, all of these templates are open source, so you are welcome to modify the entire flow if you like. By default each of the paths generate emails are to both the customer and the merchant staff.

One issue of note here, the method of contacting the merchant bank is determined by the payMethod of the order and the currently selected credit card processor. A payMethod of CC will cause WebMerchant to look at the selected credit card processor to tell it which WebDNA [include] file to call to handle the communication. So should you want to add another processor that is not currently listed, simply duplicate one of the existing ones and replace the WebDNA with the technique needed to talk to the new processor. Further, a payMethod of AC causes WebMerchant to directly call AccountAuthorizer.inc to get the Authorization. A sample WebDNA include file has been provided, and typically these are used for internal purchase order or other non-credit card purchase approvals that talk to an existing accounting system to determine the credit available for this customer.

Another example of using the hooks in the processing would be to automatically do order fulfillment of unique serial numbers for each product

purchased. The included Storebuilder templates show one way to provide this feature. Another use would be to export data to accommodate your accounting database such as QuickBooks, PeachTree, Flexware etc or the more generic databases of Oracle, Sybase, Paradox, and so forth.

ORDER MANAGEMENT

The ability to see how your business is doing is key to your survival. WebMerchant is the tool to help you do this.

WebMerchant is written as an open source set of WebDNA templates. As such, there are a variety of ways by which they can be set up and order information can be displayed. WebMerchant order reporting and information is accessed from the AdminOrders.tpl file in the WebMerchant directory. Sales reporting by default is time-based; that is, today's sales are reported first then yesterday's, then sales for the month.

Some of the modifications that can be made are, reporting for a particular item or grouping of items, the popularity of an item (i.e., what is your hottest selling product). Similarly, one may wish to report on the number of orders that had problems that were resolved and completed versus the amount and type of order problems in a given day, week, or month.

Using a Text Editor vs. HTML Editor

Designing templates involves both HTML and WebDNA. Because there are many HTML authoring tools available to make the page layout significantly easier than writing HTML tags by hand in a text editor, it is good to design the template first in an HTML editor to make it look the way you want. If there are areas such as tables to be generated by WebDNA, write out one row with dummy data just so you can see the formatting.

Because WebDNA uses square brackets rather than angle brackets to enclose its tags, you can often enter the WebDNA tags directly in the HTML editor. However, you must be careful that the square brackets are not encoded when the file is saved. You can check this by creating a new document, placing the text "[date]" in the file and saving it. Open the file in a 'raw' text editor (not an HTML editor) and see if the square brackets appear intact. If the brackets do not appear intact, you may still be able to enter WebDNA tags in the HTML editor if it has a "raw" mode where you can enter HTML it may not understand. Sometimes this data is stored without being encoded. Perform the same test mentioned above, except enter the data in

“raw” mode. As a safety precaution, close the file, open it, and save it again before checking the file in a text editor.

If the square brackets are still encoded then you will have to enter all the WebDNA tags in a text editor. This task is not as tedious as one might think because most of the formatting contained in the HTML design can be finished before resorting to a text editor. For more options, see the section on Dreamweaver integration for more about our alternative syntax for WebDNA to work with more HTML editors (this needs to be reworded).

WORKING WITH TABLES

Another problem area can be with tables. Because WebDNA can be used to create tables of variable length based upon data in a database, you will often want to include the formatting for a generic table row within a WebDNA context that lists the results of a database search. In order to do this, you need to place the WebDNA in between the table heading tag and the first row tag. This area is not a valid place for HTML, but it is a valid area for WebDNA (it is valid because by the time a visitor’s browser ‘sees’ the page, all of the WebDNA is gone and has been replaced by HTML tags). Unfortunately some HTML editors do not allow you to place any tags here (even though they will be removed before the file is sent to the browser that views the WebDNA file). Review the following example:

```
[search ...]
<table>
[founditems]<tr><td>[name]</td></tr>[/founditems]
</table>
[/search]
```

Again, if your HTML editor is not able to place tags in a particular location, you will need to open the file in a text editor and manually add the WebDNA tags.

Using Contexts vs. Commands

Because WebDNA offers multiple methods for performing certain functions, like searching a database, it is not always easy to know when to use a URL command and when to use an embedded context.

In general, using a command whenever possible is less work since you don't have to build up the search context on the result page. The input form that initiates the search is used by WebDNA to build an implied search context for you. As with most things, however, there is a trade-off between ease of use and flexibility. Fortunately though, you have the ability to do both.

Although it requires more work, it is recommended that you use contexts instead of commands. Contexts allow you to control the reaction of your template better if the visitor changes the URL by hand. Contexts also allow much more security over the data you are returning.

If you use contexts, you will still need a input form that collects all the necessary search information. The results page will then have a [Search] context using the variables submitted from the input form. For example, the input form might look like the following:

```
<form method="post" action="results.tpl">
<input name="search_data" value="">
<input type="submit" value="submit">
</form>
```

The [Search] context in the results template would then look like this:

```
[search db=databasename&eqskudata=[url][search_data][url]]
...
[/search]
```

SEARCHING

Now that you have seen some situations where commands have advantages over contexts, review the following tips on how to use contexts in a way that retains the flexibility of commands but without their inherent weakness (i.e., a command is susceptible to being modified via the URL or form data that could lead to unwanted actions).

Searching represents a context and command where the parameters can vary wildly. Oftentimes, you will want to search a single database in many

ways. The search command gives you this flexibility since you can pass different parameters based upon where the search request comes from.

For example, review the following three different hypertext links:

```
<a href="results.tpl?command=search&db=products.db&max=1
&eqCATEGORYdatarq=software&asSKUsort=1">
<a href="results.tpl?command=search&db=products.db&max=1
&woDESCRIPTIONdatarq=fast&asSKUsort=1">
<a href="results.tpl?command=search&db=products.db&max=1
&eqSKUdatarq=1106A&asSKUsort=1">
```

All three of the above hypertext links search the database in a different way. The first finds all products in the Software category, the second finds all products with “fast” in the description field, and the third finds a particular product by its SKU. Because the search parameters are not hard-coded on the results.tpl page, you have a lot of flexibility.

This flexibility contrasts with the use of an embedded [search] context on the results.tpl page.

For example, you could use a [Search] context in the following way:

```
<a href="results.tpl?category=Software">
Results.tpl:
[search db=products.db&max=1&asSKUsort=1
&eqCATEGORYdatarq=[url][category][url]]
...
[/search]
```

This reduces the size of the URL necessary to link to the results page and does not allow hackers to change the database parameter or max value as they could in the search command example. However, there isn’t a clean method (as it is currently written) to search by keyword or SKU as you could in the previous example.

Fortunately, you can achieve the same flexibility with the [Search] context, while still retaining much of its security, by simply using a different syntax. For example:

```
<a href="results.tpl?__eqCATEGORYdatarq=Software">
<a href="results.tpl?__woDESCRIPTIONdatarq=fast">
<a href="results.tpl?__eqSKUdatarq=1106A">
Results.tpl:
[search db=products.db&max=1&asSKUsort=1&[formvariables
name=__&exact=f]
[getchars start=3][name][getchars]=[url][value][url]
[/formvariables]]
[/search]
```

In the above example, all the variable search parameters are placed in the URL and all the constant search parameters (e.g., db, max, sort) are in the context. Additionally, two underscores were added in front of the search parameter names in order to make it easier to use the [formvariables] context to find just those parameters inside the context. This provides the flexibility of searching the database in many ways, but removes many unnecessary or constant parameters from the URL. It also provides secure access control to the products database more fully.

Why were two underscores chosen? First, they are unobtrusive and don't significantly change the way the parameters are named so you can still read and understand the URLs. Secondly, it is very unlikely that other parameters will include two underscores (so when we loop through the [FormVariables] that contain two underscores we can be assured that the parameters are meant for the [Search] context).

Lastly, we can always get the original parameter name by simple removing the first two characters. Note that when we add these variables to the beginning context tag, we use the [GetChars] context to remove the preceding underscores since the [Search] context doesn't understand them with underscores.

Of course, this technique can be used in many other situations than the one described here.

Replacing Commands with Contexts

Using the shopping cart commands as an example, this section describes how to replace the add/remove/showcart commands with embedded contexts. Typically, you have a single shopping cart page to display the current contexts of your cart. Using one of the shopping cart commands, all pointing at the same page, you could either add a product to the cart, remove a product from the cart, or show the contexts of the cart (using showcart).

For example:

```
<a href="shoppingcart.tpl?command=add&cart=[cart]
    &db=products.db&sku=1106A">
<a href="shoppingcart.tpl?command=remove&cart=[cart]
    &db=products.db&index=1">
<a href="shoppingcart.tpl?command=showcart&cart=[cart] &db=products.db">
```

The shopping cart page can then just have an embedded [LinItems] context to show the current contents of the cart.

You might be tempted to use the shopping cart contexts, [AddLineItem], [RemoveLineItem], and [OrderFile] to perform the same functions on three different pages: addcart.tpl, removecart.tpl, and showcart.tpl. The reason for doing this is because you didn't want the [AddLineItem] context executed when removing from the cart and so forth. Fortunately, you can use the [ShowIf] context and a custom action/command parameter to get the flexibility of commands on a single page.

Here's how:

```
<a href="shoppingcart.tpl?action=add&cart=[cart]">
<a href="shoppingcart.tpl?action=remove&cart=[cart] &index= 1">
<a href="shoppingcart.tpl?cart=[cart]"> shoppingcart.tpl:
[orderfile cart=[cart]]
[showif [action]=add]
[addlineitem cart=[cart]&db=products.db&sku=[sku]][/add lineitem]
[/showif]
[showif [action]=remove]
[removelineitem cart=[cart]&db=products.db&index=[ index]]
[/showif]
[lineitems]
[/lineitems]
[/orderfile]
```

Seemingly, all that has been changed is the parameter name from "command" to "action". However, a lot more has been done. First, the name "action" was used because it has no internal meaning in the WebDNA language (unlike the name "command", which does have a special meaning). Second, using the [ShowIf] context to add and remove items from the cart using embedded contexts provides the ability to hard-code certain parameters that don't change (such as the database value).

It is also possible to combine the technique introduced in Part 1 with this technique to create a single page that appends, deletes and replaces records from a database.

LOGGING INFORMATION

One of the most important functions of a Web site is its ability to collect and log information about the people accessing it. Web servers often have logging capabilities built-in, but the log can contain information that is not very detailed or specific to your particular needs.

Fortunately, with WebDNA and WebDNA, you can easily create detailed logs to store all the information you could possibly want about visitors to your site.

You can even have multiple logs that don't contain a lot of extraneous information.

Log files are created as simple text files using the [AppendFile] context. A very simple log will collect the IP address of every visitor to your home page. Although this information can usually be obtained from your web server log, the WebDNA log can be more useful as it ignores requests for graphic files and other information you may not want.

Adding the following WebDNA to your home page will create a custom log file for you as shown below:

```
[appendfile user.log][ipaddress]
[/appendfile]
```

The file "user.log" will have the IP address (followed by a carriage return) written out for every visitor to the page that contains the WebDNA mentioned. You should see how you can easily create a custom log that stores all the search information coming from a particular form as well.

Logs are usually created for use as databases to other WebDNA functions. This lets you evaluate the data you collect much more easily. As you learn more about WebDNA you may wonder why WebDNA database commands were not chosen to add information to the log (the [AppendFile] context simply writes text to the end of a file). The reason is speed. Writing text to the end of a file is much faster and more RAM efficient, than reading a database and adding records to it. Because the log files/databases can be very large, small improvements in speed and RAM requirements make a big difference when they are implemented.

Chapter 2 – WebDNA Tutorial

This chapter provides information and instruction on the following issues:

- . WebDNA Theory of Operation
- . Conceptual Development of the TeaRoom; a web site for a retail business

WebDNA Theory of Operation



Figure 4. WebDNA/Web Server Interaction

WHAT IS WEBDNA

WebDNA is literally a text processing engine that responds to incoming web browser requests from a Web Server, as shown in Figure 4. A browser request originates from the user in the form of

- . Hitting a web page
- . Performing a query for information, such as a keyword search, or
- . Performing a submit action to retrieve information, such as a quote on an item, or a subtotal on an added item in a shopping cart.

Prior to a request, WebDNA exists in a waiting state.

HOW WEBDNA ACTS ON A WEB SERVER REQUEST

From WebDNA's perspective, an incoming web server request is a request to merge the text in the template with the incoming information, similar to the earlier computing days when a form letter drafted in a word processor was to be merged with a list of names to which the letter would be sent. For example, using a template to put in today's date, in WebDNA, the variable [date] is used to substitute an incoming date request on an html form. The WebDNA date tag then reads the system clock for today's date and replaces the [date] text in the template with the current date. Similarly, WebDNA would process a search request from a Web Server for a name, say John Smith. This name would be passed as a form variable 'name = John Smith'. The template that the browser is hitting would have full use of the form variable [name] and it would be replaced with John Smith wherever that is used in the template. So, if the function of the template is to search a database for the name John Smith it would look something like this in the template:

```
[search db=people.db&eqPERSONdatarq=[name]]
[founditems]
[PERSON]<BR>
[ADDRESS]<BR>
<BR>
[/founditems]
[/search]
```

Given a database, people.db that has the fields of person and address, the above would give you a list in your HTML of those matching records showing the name of the person on one line and the address on the next, then a blank line between each record.

As another example, to add text to the end of an arbitrary text file, put an [AppendFile] context into a template. AppendFile creates a new file if one does not exist already. All text is put at the end of the file, after any data that may already be there.

Note: AppendFile does not 'understand' databases. If you want to append a new record to the end of a database, use [Append](#) instead.

```
[AppendFile file=SomeTextFile]Hello, my name is Grant. The time is [time]
```

```
This is a second line[/AppendFile]
```

The text file "SomeTextFile" opens, and the text
Hello, my name is Grant. The time is 13:43:01. This is a second line.

is written at the end of the file. Notice that carriage returns inside the context are written to the file exactly as they appear. Also notice that any WebDNA [xxx] tags inside the context are substituted for their real values before being written to the file.

Security Note: By default, all files created by WebDNA on Macintosh and UNIX web servers are tagged with a special code or file permission telling your web server not to display them via URL. If you want files to be visible to outside browsers, use the optional settings below. To further protect your files, refer to the security section.

Parameter	Description
secure	“T” for files that should be secure—web server will not display them “F” for files that should be visible via URL—web server will display them Example: [WriteFile secure=F&file=SomeFile]...[/WriteFile]
file	The name (or relative path) of the file to create.

PREPARING YOUR SITE FOR USE WITH WEBDNA

When designing your web site, there is no particular order of what is needed first from WebDNA’s point of view. You can set the form, the look and feel of the web site, before building the functionality that will be used to showcase your products and transact with a customer’s shopping cart to process an order. Alternatively, you can first put in the functionality you desire from the WebDNA templates and lay out the form of your web site.

However, from an order flow point of view, it is good to think of your site in the same way one might think of a fax conversation. For example, if you receive a one-page order form from a company by fax, that page may have all available items for sale. You can check the quantity desired for each listed item and select all desired items and assign a cost for each item listed. There might even be a place to add the appropriate sales tax for the area you are located. All this information can be filled out but the vendor will not know of your order until you fax it back to them. So you do just that.

You were not able to figure out the shipping and handling since this field was not included on the form for good reason; shipping costs vary by location and the vendor does not want to burden you with that detail. Instead, the vendor receives the order, logs it in to their system, and then receives the shipping and handling costs back from the system based on the shipping address you

provided. The vendor then faxes the confirmation of your order with the added shipping and handling costs back to you for your approval. You look over the order information and accept the final figures. You fax back the approved order and the credit card information you provided with the order form is used to bill you for the order.

In the same way as the fax example described above, WebDNA will act upon requests received from the browser. In order to receive a request, the user will fill out the order form and submit the order to your Web Server. In turn, the Web Server moves the order request to WebDNA where the request is processed. The processing is performed with WebDNA code.

REQUEST PROCESSING WITH WEBDNA

As indicated in Chapter 1, a WebDNA command directs WebDNA to perform an action that fulfills an incoming web server request and responds to the web server with the results from the interpreted template. In this context, WebDNA does what it is told to do; such as:

- . return the results of a database search
- . return detailed information to be displayed to satisfy the client request
- . return updated price information for an amended order
- . or anything else you can think of.

WebDNA's behavior can be thought of in the same way a person in the US Army behaves; they act upon and follow orders to the letter; doing what they are told in the exact way in which they are told. Therefore, it is important to realize that when you don't get what you expect in processing, the source of the error in what you are telling it to do will usually be found outside of WebDNA. For example, if an unprintable character is sent to WebDNA, an otherwise expected printed character will not display. Some characters may be printable on a Macintosh machine but not on a Windows machine. Similarly, a Macintosh machine uses a carriage return character only at the end of each line. A Windows machine uses a carriage return and a line feed. Formatting problems will occur between platforms causing display and printing problems to some users. Therefore, it is important to consider the behavior of text between different computer platforms.

Development of the TeaRoom Database

This part of the WebDNA tutorial is intended to help you learn how to set up your own web commerce site using the TeaRoom templates included in your WebDNA package. The tutorial guides you through the creation of an imaginary eCommerce site, called the *TeaRoom*, which allows visitors to:

- . Search for products in a database.
- . Choose the ones they want from the list of found items.
- . Purchase the products they have chosen over the Internet.

While you are learning how the site is constructed, you will learn a variety of WebDNA concepts and issues laying the foundation for more advanced concepts and techniques. The WebDNA code provided in this example shows an alternative syntax from the [xxx] square-bracket style shown elsewhere in this manual. This syntax is called 'XML' syntax because it more closely resembles the style of tags that newer graphical HTML editors (such as GoLive, CyberStudio and Macromedia DreamWeaver) can understand. WebDNA works with both the 'classic' syntax and the newer XML syntax. The newer syntax is provided mainly to help graphical HTML editors display and edit WebDNA templates without their annoying habit of destroying embedded square brackets in your WebDNA code. Because the new syntax looks so much like HTML, editors tend to leave it alone rather than rewriting it into what they 'think' is proper HTML.

Classic syntax: [include file=fred.inc&raw=t]

XML syntax; <DNA_include file="fred.inc" raw="t">

The best method for going through the TeaRoom tutorial is to run your web server software and browser on your web site development system, then play the role of a shopper as you study the explanations of the HTML and WebDNA that are used to create the site.

TUTORIAL: OVERVIEW

Tom and his wife Katy live in southeastern Vermont. Katy runs a tearoom and gift shop. Tom supplements his income as the pastor of a small church by substitute teaching and Internet consulting. He has already created a successful commercial site for a software company, but customers must email or call an 800 phone number to place their orders.

Because Katy's business is seasonal and relies substantially on tourism, Tom and Katy have decided to make some of the teas and gifts that the shop sells available to Internet customers. Tom also thinks that his software company might be interested in reducing their 800 number calls and shipping expenses by accepting and delivering orders with the WebDNA solution.

THE TEAROOM DATABASE

Tom has created a database of the products they are going to sell in a database program and exported it as a tab-delimited text file. Because his database program did not export the field names along with the data, he has added the database field names as the first record in his database.

The following example shows the field names and the first record of the database. "<tab>" represents the tab character required between each field, and "<cr>" represents the carriage return required at the end of every record.

```
SKU<tab>Title<tab>Description1<tab>price<tab>taxable<tab>
CanEmail<tab>UnitShipCost<tab>UnitShipWeight<tab>
Ingredients<tab>Description2<tab>Description3<tab>
Category<tab>CatPageNum<tab>HasPhoto<tab>HTMLFontColor<tab>
PhotoName<cr>
```

```
10001<tab>All Day Breakfast Tea<tab>Keemun, a small and robust leaf, is only
cultivated in Anhui province, a region of Southern China where the
mountains are covered with teabushes. With its haunting flavor and sweet
aroma, this special leaf has come to be known in the trade as "orchid"
Keemun.<tab>10.00<tab>T<tab> F<tab>0 <tab>0<tab>Finest China
Keemun and Silver-Tip Formosan Oolong Tea Leaves<tab>Beyond English
Breakfast<tab>World's Finest Breakfast Tea Leaves<tab>Black
Teas<tab>9<tab>T<tab><tab> 10001.jpg<cr>...
```

STEP 1: ENTERING THE SITE

The entry point to our on-line store is the first page. To get there, type the following URL into the location field of your browser:

```
http://< your server >/WebCatalog/TeaRoomXML/Entry.tpl
```

Your browser will display the TeaRoom home page, as shown in Figure 5.

Most of the time when entering a real store, you expect that the storekeeper will have organized the displays into various departments, sections or categories. You are going to do the same thing with the first page of your on-line store.

You are also going to learn how to make your commerce site creation activities more efficient by using WebDNA's <DNA_Include> tag to add elements repeated on all of your templates such as logos and addresses.

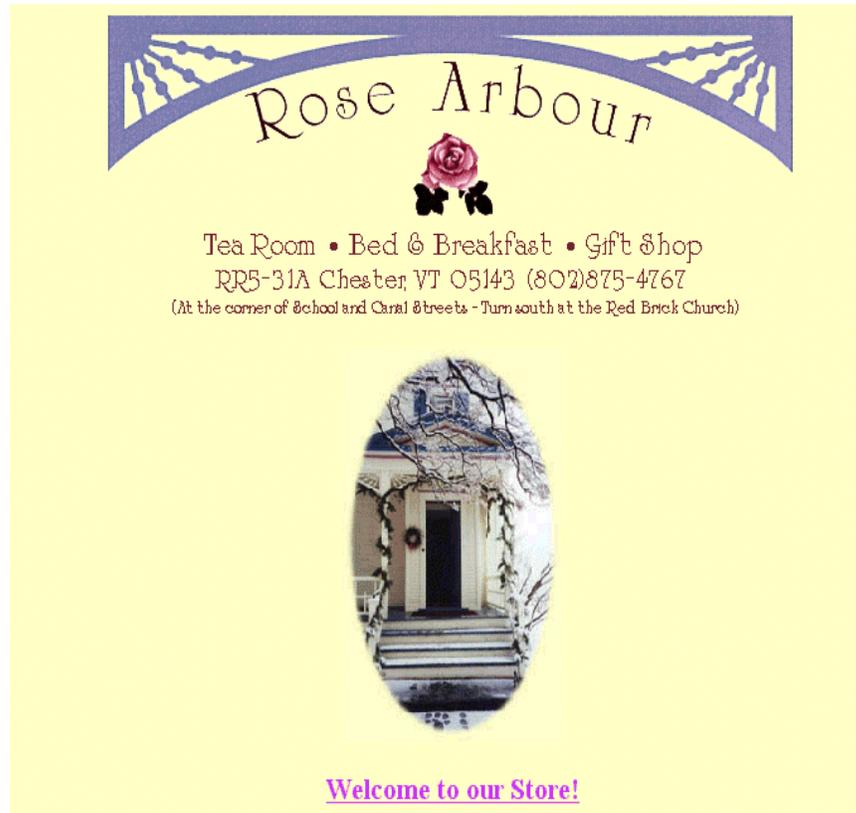


Figure 5. TeaRoom Home Page

Review the following description of the Entry.tpl page shown in the prior HTML page sample.

Entry.tpl

```
01| <!--HAS_WEBDNA_TAGS_XML-->
02| <HTML>
03| <HEAD>
04| <TITLE>Welcome To Rose Arbour - Come on In!</TITLE>
```

```

05| </HEAD>
06|
07| <DNA_Include file="TeaRoom_Header.inc">
08| <P>
09| <CENTER>
10| <IMG SRC="<DNA_include
    file="ImagePath.inc">TeaRoomImages/WinterPorch.jpg" WIDTH="175"
    HEIGHT="281">
11|
12| <H1><a href="Search.tpl">Welcome to our Store!</a></H1>
13| </CENTER>
14| </BODY>
15| </HTML>

```

The suffix for this page has been set to ".tpl". When the web server detects this suffix, it knows that the page should be processed by WebDNA before it is returned to the visitor's browser.

Line 01| tells WebDNA the page contains WebDNA and should be processed before being returned to the browser, and that this template is in the XML - style syntax.

Lines 02| to 06| are all pretty much standard HTML. They define the <HEAD> section of the page and the title that will be displayed by the browser in the page's title bar.

Line 07| uses the WebDNA <DNA_Include> tag to instruct the browser to use the text in a file called "TeaRoom_Header.txt" as if it were an integral part of the HTML of this page. (There could be WebDNA tags in this file as well and they would also be interpreted just as if they were part of the page.) The file includes the following HTML:

```

<BODY BGCOLOR="#FFFFCC" TEXT="#003399" LINK="#CC33FF">
<CENTER>
<IMG SRC="TeaRoomImages/RA_Title.GIF"
WIDTH="547" HEIGHT="144">
<IMG SRC="TeaRoomImages/RA_Address.GIF"
WIDTH="468" HEIGHT="72"></CENTER>

```

Because every page in this on-line store should contain the information defined by this HTML (It simply sets the background, text, and link colors and includes images of our store logo and address.), include this file on every template to save the effort of having to type it each time you create a

template. Lines 08| and 9| are also ordinary HTML including and centering an enticing image inviting people into our store.

Line 12 actually does something WebDNA related. The hypertext link beginning with "<A HREF" requests the web server to serve the page titled "Search.tpl".

12| Welcome to our Store!

The <A HREF> hypertext link instructs the web server to return a page named "Search.tpl". since the name of the requested page ends in the suffix ".tpl" and the web server has been set to treat requests for pages ending in ".tpl".

STEP 2: SHOPPING FOR PRODUCTS BY CATEGORY

It is now time for the customer (you) to enter the store. Click on the **Welcome to Our Store** hypertext link and your browser will display the following page, as shown in Figure 6.



Figure 6. TeaRoom Product List

The HTML and WebDNA for the Search.tpl page is displayed as:

```
01| <!--HAS_WEBDNA_TAGS_XML-->
02| <html>
03| <head>
04| <title>Let's Go Shopping!</title>
05| </head>
06|
07| <DNA_Include file="TeaRoom_Header.inc">
08|
09| <BR>
10| <CENTER><font size=+3>Thank You for Coming In!</font><br>
11| <hr width=300>
12| <b>Welcome to Rose Arbour's online store.</b> <BR>
13| We are committed to providing you with<BR>
14| the finest in teas, tea accessories, and gifts.
15| <hr width=300>
16| <font size=+2>Product Categories</font><BR>
17|
18| <DNA_search db="TeaRoom.db" geSKUdata="0" Categorysum="t"
19|   asCategorysort="1">
20| <DNA_foundItems>
21| <a
22|   href="Results.tpl?cart=<DNA_cart>&category=<DNA_url><DNA_Catego
23|   ry></DNA_url>&startat=1"><DNA_category></a><BR>
24| </DNA_foundItems>
25| </DNA_search>
26|
27| <hr width=300>
28| </CENTER>
29| </BODY>
30| </HTML>
```

Lines 01| to 13| are straightforward HTML including only the old friends <!--HAS_WEBDNA_TAGS_XML--> and the <DNA_Include> tag for the logo and address.

Lines 10| to 17| add and format some text ahead of the list of product categories.

Lines 18| to 22| contain the WebDNA tags to display and format, and a search for all the categories listed in the database. The returned results are also wrapped in WebDNA tags letting us click on each of the items in the category list to search for and return a list of the products in the product database belonging to that category.

The search context has several parameters, all of which are defined as parameters listed after the context name. Their purpose, in this case, is to do the following:

- . Search for all the products in the on-line store's TeaRoom.db database.
- . Summarize what was found by the product categories defined in the product database.
- . Sort the found categories in ascending order so they will be displayed alphabetically on the returned template, Search.tpl.

Further analysis of these parameters can show you what they are actually doing. The geSKUdata parameter instructs WebDNA to search the SKU field in the TeaRoom.db database for any data with a value "ge" (greater than or equal to) 0. This assumes that no product in a real database would have an SKU of 0.

The Categorysumm=T parameter tells WebDNA to search all the records in the database and to summarize every unique value it finds in the Category field. When it displays the results of the summary, it will only show the unique values in the Category field. Because every product's Category field has an identical entry for that category, the resulting display will be a simple list of the categories in our TeaRoom.db database.

However, because most of us are used to seeing lists organized either numerically or alphabetically, the following was used: Categorysort=1 and the Categorysdir=as (where "sort=1" is the sort order when sorting on multiple fields and "sdir=as" is the sort direction) to instruct WebDNA to sort the Categories found in ascending alphabetical order.

Lines 19] and 21] define a <DNA_foundItems> context. WebDNA will display the data from the records found by the search. Notice within the <DNA_foundItems> context is the tag <DNA_Category>. The results of the summarized search will therefore be a list of the product categories in the database.

However, the <DNA_Category> fieldname tag has been wrapped in an <A HREF> hypertext link enabling the visitor to conduct a further search through the on-line store.

The first part of the search –

```
<A HREF="Results.tpl?cart=<DNA_cart>
```

```
&category=<DNA_url><DNA_Category></DNA_url>&startat=1
```

starts the hypertext link, defines the template to be returned. The remaining parameters are new; each is discussed below.

```
cart=<DNA_cart>&Category=<DNA_url><DNA_Category></DNA_url>&startat=1
```

The first search parameter, `cart=<DNA_cart>`, tells WebDNA to create a shopping cart token with a unique value to be passed from page to page as the visitor moves through the site. WebDNA's "SmartCart" feature automatically creates a shopping cart token if it detects that one has not been created on any page you display that includes the `<DNA_cart>` tag. Simply linking to a page with `<DNA_cart>` tags makes WebDNA create a new token because WebDNA was invoked through the ".tpl" suffix. Important: WebDNA must have a cart token in order to keep track of visitors as they move through the site.

With the second parameter, "Category", the `<DNA_Category>` fieldname tag was enclosed within a `<DNA_url>` context. WebDNA will substitute the value found in the previous search for the `<DNA_Category>` fieldname tag but will format it as a proper URL before including it in the search to be conducted. This is necessary in case the value of the Category field has a space or other character not valid in URL's.

Lastly, the "startat" parameter tells the `<DNA_search>` context on the Results.tpl page which group of items, in the category selected, to list first (in case there are more items in the category than can be listed at one time).

The fieldname tag, `<DNA_Category>`, following the search parameters is the actual hypertext link which is then closed off by the `` tag.

To see how this page works, click on the first hypertext in the list of categories, Black Teas. Your browser will display the Results.tpl page, as shown in Figure 7.

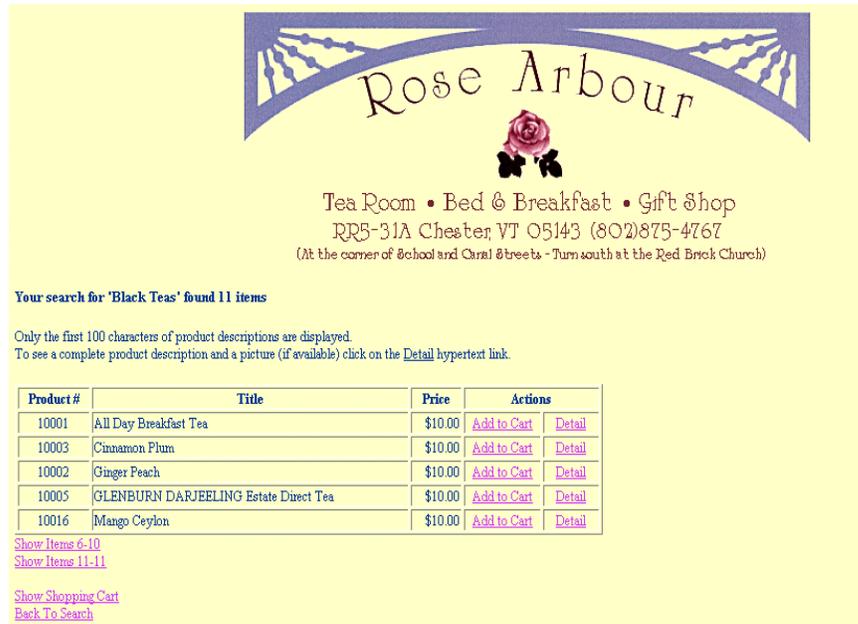


Figure 7. Black Teas Category Results Page

STEP 3: ADDING ITEMS TO THE SHOPPING CART

The Results tpl template is designed to display all the products of a particular category in an easy-to-read table. It also allows the visitor to examine the details of a product more completely and to select individual items in the list to include in a virtual shopping cart. Before they “check out” on the final order page, they will be able to define the quantities of each item they want to buy or remove items they don’t want to buy from the cart.

```
01| <!--HAS_WEBDNA_TAGS-->
02| <HTML>
03| <HEAD>
04| <TITLE>Results of Your Search for <DNA_category></TITLE>
05| </HEAD>
06| <DNA_Include file="TeaRoom_Header.inc">
```

As usual, the first six lines are set up in the first part of our HTML document. However, a little twist has been added. In line 04|, the variable name category has been enclosed in WebDNA brackets. When the browser

constructs the title for this page, it will substitute the value of 'category' that was determined when the visitor clicked on one of the categories listed on the Search.tpl page. Thus, if the visitor clicked on the category "Black Teas" in the list, the title of the returned Results.tpl page will be "Results of Your Search for Black Teas".

```
07| <DNA_search db="TeaRoom.db"
eqCategorydata="<DNA_url><DNA_Category></DNA_url>" asTitlesort="1"
asSKUsort="2" max="5" startat="<DNA_url><DNA_startat></DNA_url>">
08| <H3>Your search for '<DNA_Category>' found <DNA_numFound>
items</H3>
09| <P>
10| Only the first 100 characters of product descriptions are displayed.<BR>
11| To see a complete product description and a picture (if available) click on
the <U>Detail</U> hypertext link.<P>
```

Lines 07| to 11| set up a search using an embedded <DNA_search> context using the category determined on the Search.tpl page. Review the parameters used in the <DNA_search> context.

eqCategorydata tells WebDNA to search the database for data in the Category field equal to the search criteria following the "=" sign. By default, if you don't specify that certain fields are required to match, they are "or-ed" together, which in this example would mean, "Find all records whose Category is Black Teas or Green Teas or Tea Pots, etc.". To require the Category field to match during the search, the letters "rq" were placed after the data specifier. For a more complete explanation of "And" vs. "Or" in searching, refer to the reference later in this manual under Searching - "And" vs. "Or".

The remainder of line 07| defines how the results of this new search will be sorted.

```
asTitlesort="1" asSKUsort="2" max="5"
startat="<DNA_url><DNA_startat></DNA_url>"
```

The final parameters instruct WebDNA to sort the results of the search you are about to conduct first by product Title (Titlesort=1) and then by SKU (SKUsort=2). In both sorts the sort direction is in ascending order (Titlesdir=as) and (SKUdir=as). Finally, max=5, sets the number of records to be displayed at one time. If more than 5 products are found in a particular category, the startat parameter determines which block of 5 records to display (see <DNA_shownext> in the following example for more information).

```
12| <TABLE BORDER=1 CELLPADDING=1>
```

```

13| <TR>
14| <TH>Product #</TH><TH>Title</TH><TH>Price</TH>
<TH COLSPAN =2 ALIGN="CENTER">Actions</TH>
15| </TR>
16| <DNA_foundItems>
17| <TR>
18| <TD WIDTH=65 ALIGN="CENTER"><DNA_SKU></TD>
19| <TD WIDTH=300><DNA_title></TD>
20| <TD WIDTH=45 ALIGN="RIGHT"> $<DNA_format.2f><DNA_
price></ DNA_format></TD>
21| <TD WIDTH=70 ALIGN="CENTER"><A HREF="ShoppingCart.tpl
?command=add&cart=<DNA_ cart>&db=TeaRoom.db&sku=<DNA_
url> <DNA_ sku></DNA_ url>">Add to Cart</A></TD>
22| <TD WIDTH=50 ALIGN="CENTER"><A
HREF="Detail.tpl?cart=<DNA_ cart>&sku=<DNA_ url> <DNA_ sku>
</DNA_ url>">Detail</A></TD>
23| </TR>
24| </DNA_ foundItems>
25| </TABLE>
26| <DNA_ shownext>
27| <A HREF="Results.tpl?cart=<DNA_ cart>&category=<DNA_
url><DNA_ category></DNA_ url>&startat=<DNA_ start">
Show Items <DNA_ start>-<DNA_ end></A><BR>
28| </DNA_ shownext>
29| <BR>
30| </DNA_ search>
31| <A HREF="ShoppingCart.tpl
?command=showcart&db=TeaRoom.db&cart=<DNA_ cart">"> Show
Shopping Cart</A><BR>
32| <A HREF="Search.tpl?cart=<DNA_ cart">">Back To
Search</A><BR>
33| </BODY>
34| </HTML>

```

Lines 12| to 30| define how the records found by the search will be displayed in the table. The HTML in lines 12| through 15| are straightforward HTML tags setting up the table headings.

Lines 16| and 24| open and close a <DNA_ FoundItems> context within which are displayed all of the WebDNA tags with their associated data as returned from the search, and which you want to display. Some pieces of this data are simply displayed for the visitor's information. However, other returned data is wrapped within <A HREF> tags that enable you to act upon it.

Lines 18| and 19| WebDNA simply substitutes the value of the SKU and Title for each record found. Line 20| does the same, but is enclosed in the <DNA_Price> fieldname tag within a <DNA_Format> context. The <DNA_format .2f> tag tells WebDNA to display the data in Price field as a decimal number (The “f” means floating point.) with 2 places following the “.” (decimal point).

Lines 21| and 22| are the workhorses on this page. Analyze their parts further.

```
21| <TD WIDTH=70 ALIGN="CENTER"><A HREF="ShoppingCart.tpl
    ?command=add&cart=<DNA_cart>&db=TeaRoom.db&sku=<DNA_
    url> <DNA_sku></DNA_url">Add to Cart</A></TD>
```

The first part of line 21| sets the column width for this cell and horizontally aligns its data in the center. The second part begins the hypertext link, instructs WebDNA to return the page ShoppingCart.tpl and sends the add command which tells WebDNA to create a shopping cart or order file and add whatever is defined by the parameters after the “?” to the shopping cart file designated by the cart=<DNA_Cart> tag. (Shopping cart files become order files when the visitor finalizes their purchase with the purchase command.)

You should note that you could also use the <DNA_addlineitem> context on the resulting shopping cart page in order to add the product to the cart. In general, you should use contexts instead of commands to perform WebDNA functions since they offer many advantages. However, the add command is used here so you can learn how to use it. The third part of the link designates which database is to supply the item being added, db=TeaRoom.db, and which item in the database is the one the shopper has chosen, sku=<DNA_URL><DNA_SKU></DNA_URL>. Just in case the data in the field SKU has any “illegal” characters (spaces, for instance) the <DNA_SKU> fieldname tag is wrapped in a <DNA_URL> context.

In the next section you’ll see what the ShoppingCart.tpl template does with the item you asked WebDNA to add to this visitor’s shopping cart.

```
22| <TD WIDTH=50 ALIGN="CENTER"><A HREF="Detail.tpl
    ?cart=<DNA_cart>>&sku=<DNA_url><DNA_
    sku><</DNA_url">Detail</A></TD>
```

Line 22| By this time, you should be able to read line 22| and describe what it does, but just in case you need a little help, it is analyzed further below.

The first part of line 22| once again sets the column width for this cell and

horizontally aligns its data in the center. The second part begins the hypertext link, instructs WebDNA to return the page Detail.tpl and sends the value of the cart and SKU to look up in the TeaRoom database. An embedded search in the Detail.tpl page will find the information for the SKU value given. Of course, to be safe the <DNA_SKU> tag is wrapped in a <DNA_URL> context. When you examine the Detail.tpl template you'll see how the information is displayed.

Lines 24| and 25| close out the <DNA_FoundItems> context and the <TABLE> tag.

Remember the parameter max=5 is included in the search context. Line 26| is a <DNA_ShowNext> context which encloses an <A HREF> hypertext link and that does the following things.

```
26| <DNA_shownext>
27| <A HREF="Results.tpl?cart=<DNA_cart>&category=<DNA_url>
   <DNA_category></DNA_url>&startat=<DNA_start">
   Show Items <DNA_start>-<DNA_end></A><BR>
28| </DNA_shownext>
```

Part one, <DNA_ShowNext><A HREF="Results.tpl?cart=<DNA_cart>&, opens the <DNA_ShowNext> context, and starts the hypertext link that will allow people to view the items found from the search but not part of the first 5 items.

Part two, category=<DNA_url><DNA_category></DNA_url>&startat=<DNA_start>, defines the search parameter chosen on the Search.tpl page. The value of the startat parameter, <DNA_Start>, is a value automatically determined by the <DNA_shownext> context. For example, if there are 23 items found in a category, and you are displaying 5 records at a time, the <DNA_ShowNext> context will loop 4 times so you can build up links for all the additional groups of 5 products: 6-10, 11-15, 16-20, 21-23. For each loop through the WebDNA contained in the <DNA_shownext> context, there are two values available - <DNA_Start> and <DNA_End>. In the example, the start values are 6, 11, 16, 21, and the end values are 10, 15, 20, and 23.

Lines 31| and 32| are site navigation links. Line 31| creates a link telling WebDNA to return the template ShoppingCart.tpl, and show all the items currently in the cart.

Line 32| is an exact duplicate of the hypertext link on the Entry.tpl page and will return the visitor to the Search.tpl page to search for products in the store by category.

As the customer, you have decided you would like to purchase the Cinnamon Plum black tea. Click on the Add to Cart hypertext link in the table row for Cinnamon Plum. Your browser will display the ShoppingCart.tpl page. It will list one product, Cinnamon Plum tea, with an input field for you to set the quantity of tea canisters you would like to buy. Review the following section to see how this page works.

STEP 4: USING THE SHOPPING CART PAGE

There are two circumstances when a visitor to this site will view the shopping cart page, ShoppingCart.tpl. The first is whenever they click on any Add to Cart link on the Results.tpl page, and the second is when they click on any link that takes them directly to the page using the showcarts command in order to view the current contents of their cart.

Analyze the ShoppingCart.tpl template and see how it begins the process of turning the WebDNA site, which up to now has been mostly finding and displaying records in a database, into a Web commerce site that lets your visitors make personal purchasing decisions and finalize them over the web. That includes choosing products and quantities, choosing preferred shipping and payment methods, having all their individual charges calculated and their final purchase decision acknowledged with a "Thank You!" on the web and by personal email.

By this time, you should have little difficulty describing what lines 01| through 10| are doing. Even the hypertext link in line 08| should be familiar to you. It is used on every page except the Search.tpl page that is the one it links to lines 09| and 10| are merely instructions to the visitor on how to proceed and include no WebDNA tags.

```
01| <!--HAS_WEBDNA_TAGS-->
02| <HTML>
03| <HEAD>
04| <TITLE>Your Shopping Cart</TITLE>
06| <DNA_Include TeaRoom_Header.txt>
07| <H1>Shopping Cart</H1>
08| You have just added an item to your shopping cart. From here, you
    may press your browser's "Go Back" button to see the page you just
    came from, or you can <A HREF="Search.tpl?cart=<DNA_cart">">
go to the Category Search page.</A><P>
09| If you are done shopping, please fill in the quantity for each product
    you would like to buy, choose a shipping method and ship to
    destination, and then click "Proceed to Final Checkout".<br>
10| <H3>Your Shopping Cart Contains the Following Items</H3>
```

On every page until now, parameters have been passed to WebDNA from within <A HREF> hypertext links. On this page, an HTML form will be used to accomplish the same thing.

A form is required because you are asking the visitor to provide input on several things including the quantity of each product they want to buy, the shipping method they want us to use, and the State to which they want us to send their purchase. All of this input will be used by the invoice page to calculate extended prices, tax rates and totals, shipping costs, and the grand total for all the appropriate charges.

```
11| <FORM METHOD="POST" ACTION="Invoice.tpl">
12| <INPUT TYPE="HIDDEN" NAME="cart" value="<DNA_cart">">
```

Lines 11| and 12| define the method the form should use and the action to be carried out using the form information. In line 11| WebDNA is instructed to return the page Invoice.tpl. Line 12| begins the list of parameters passed to the invoice page. These are the parameters you have seen listed after the '?' in all the hypertext links used up to now. We're using hidden input fields for which values have been predefined because no visitor input is required in this case. Line 12| defines the value of the cart.

```
13| <TABLE BORDER="1" CELLPADDING=1>
14| <TR ALIGN="CENTER">
15| <TH WIDTH="38">Line #</TH>
16| <TH WIDTH="65">Product #</TH>
17| <TH WIDTH="250">Item Title</TH>
18| <TH WIDTH="30">Qty</TH>
19| <TH WIDTH="45">Price</TH>
20| <TH WIDTH="70">Put It Back!</TH>
21| </TR>
```

Lines 13| through 21| set up the table and table headings you will use to display the items the shopper has chosen on the ShoppingCart.tpl page.

22| <DNA_lineltems>

```
23| <TR>
24| <TD ALIGN="CENTER" WIDTH="38"> <DNA_linelIndex></TD>
25| <TD ALIGN="CENTER" WIDTH="65"> <DNA_SKU></TD>
26| <TD WIDTH="250"> <DNA_title></TD>
27| <TD WIDTH="30"> <INPUT TYPE="TEXT" NAME="quantity<DNA_
linelIndex>"SIZE="3" VALUE="<DNA_quantity">"></TD>
28| <TD ALIGN="RIGHT" WIDTH="65"> <DNA_format 6.2f> <DNA_
price> </DNA_format></TD>
```

```

29| <TD ALIGN="CENTER" WIDTH="75"><A
    HREF="ShoppingCart.tpl?command=remove&db=TeaRoom.db&ind
    ex=<DNA_lineIndex>&cart=<DNA_cart">">Remove Item</A></TD>
30| </TR>
31| </DNA_lineItems>
32| </TABLE><P>

```

The next few lines might have a ring a familiarity to them. A little examination suggests they are very much like the HTML and WebDNA used earlier to display the results of your product search. However, rather than use a `<DNA_FoundItems>` context a `<DNA_LineItems>` context is used. A `<DNA_LineItems>` context lists the contents of the shopping cart file that the visitor created when they clicked on the Add to Cart link in the product list displayed by Results.tpl. For a more comprehensive discussion of just what makes up the shopping cart file, refer to the reference section and the file formats section of this manual.

Lines 22| and 31| open and close the context. Line 24| introduces the `<DNA_LineIndex>` a tag, a tag available only inside the `<DNA_LineItems>` context. As each item is added to the shopping cart file a new line is created and given a number. Numbering begins with one and is incremented by one each time a new item is added. In line 24| the `<DNA_LineIndex>` tag is used to number each row containing a shopping cart item. Lines 25| and 26| contain the SKU and product title for each item in the shopping cart.

Line 27| incorporates an HTML text input field, the first opportunity the shopper has to input information for which you will need to calculate extended prices and totals. Notice the use of the `<DNA_lineindex>` tag to name the field. The quantity for each item submitted at the final purchase will have a unique name when the form is submitted, i.e. "quantity1", "quantity2", "quantity3", etc. Its value will be the number entered by the shopper.

```

27| <TD WIDTH="30">
    <INPUT TYPE="TEXT" NAME="quantity<DNA_lineIndex>"
    SIZE="3" VALUE="<DNA_quantity">"></TD>

```

Lines 28| and 29| should look familiar. With one exception, they're identical to the Price and Add to Cart lines used on the last page . . . with one exception. Rather than use the add command in the hypertext link, the remove command was used. It does just what you would expect it to do; it removes the item referenced by the SKU from the shopping cart file, re-indexes the lines in the shopping cart, and returns the ShoppingCart.tpl template with the altered file information.

```

33| <CENTER>Shipping Method: <SELECT NAME="__shipVia">
34| <OPTION VALUE="UPS" <DNA_showif<DNA_
   shipVia>=UPS>selected </DNA_showif>>UPS Ground
35| <OPTION VALUE="FedEx 2 Day" <DNA_showif<DNA_
   shipVia>=FedEx 2 Day>selected </DNA_showif>>FedEx Two-Day
36| <OPTION VALUE="FedEx 1 Day" <DNA_showif<DNA_
   shipVia>=FedEx 1 Day>selected </DNA_showif>>FedEx Overnight
37| </SELECT><P>
38| Ship to: <INPUT TYPE="radio" NAME="__shipToState" VALUE="CA"
   checked> California <INPUT TYPE="radio" NAME="__shipToState"
   VALUE="VT"> Vermont <INPUT TYPE="radio"
   NAME="__shipToState" VALUE="Other"> Other (Required for
   further processing)<P>
39| <INPUT TYPE="SUBMIT" VALUE="Proceed to Final
   Checkout"></CENTER>
40| </FORM>
41| </BODY>
42| </HTML>

```

Lines 33| to 37| provide another opportunity for the shopper to provide input needed to make the final shipping cost and grand total calculations. These lines define an HTML pop-up menu form element. However, another little WebDNA twist has been added to it. By wrapping each selected option in a `<DNA_ShowIf>` context so that the proper pop-up item will be selected if you come back to this page after making a selection. This technique is very useful in many situations. The reason the names of the input fields are preceded by two underscore characters will be made clear on the invoice page.

Line 38| adds another form element to the page. It's an HTML radio button form element asking the user to tell you whether their order is to be shipped to California (CA), Vermont (VT), or Other. Katy's parents ship some of the items from California so they have to collect sales tax for those shipments, and Katy has to collect sales tax for shipments to Vermont, but sales tax will not be collected for shipments to other states. You will use the shopper's choice to determine whether sales tax will be added to the final invoice and what sales tax rate in the `TaxRates.db` database will be used to calculate the tax. More about how this happens when you examine the `Invoice.tpl` template.

Line 39| adds the final form element to the page, a submit button that sends all the form information to WebDNA for processing and returns a completed "trial" invoice on which all the necessary extended price, shipping cost, tax totals, and grand totals have been made.

By now, the power and flexibility that WebDNA brings to developing a web commerce site should be pretty clear.

While you were looking at the brief descriptions of teas listed on the Results.tpl page you may have been intrigued by the GLENBURN DARJEELING tea. However, the description was probably too brief for you to decide whether you wanted to add it to you shopping cart. Click on the Back button of your browser to return to the list of black teas. The right most column of each row in the list has a Detail hypertext link. Click on the Detail hypertext link for GLENBURN DARJEELING. Your browser will display the Detail.tpl page (i.e., Rose Arbour's product detail page), as shown in Figure 8.

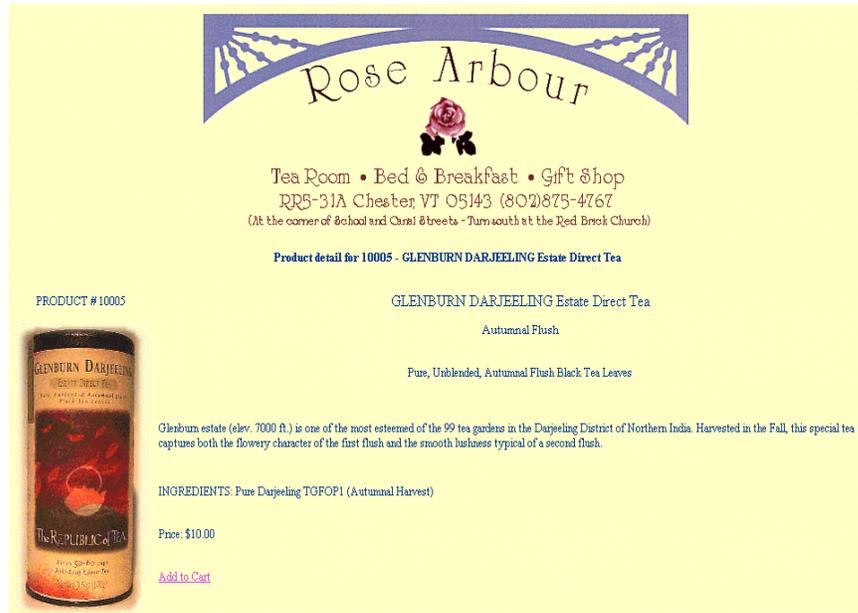


Figure 8. Detail Page for GLENBURN DARJEELING Tea

STEP 5: USING THE PRODUCT DETAIL PAGE

All the searches we have allowed our shoppers to make have returned lists of products with their SKUs, their Titles, and perhaps some text describing the product. Unless they already know something about the products in the list, it is likely that they will want to see more information before they buy.

The Product Detail page can give them the additional information they want, including a picture if it is available. Here is the HTML/WebDNA for the Product Detail page. This page is returned when the shopper clicks on the Detail hypertext link on the Results.tpl template.

Most of the lines on this template do almost exactly the same thing that the lines on the Results.tpl template do. There are some fieldname tags like <DNA_SKU> and <DNA_Title>, and there is a <DNA_Search> context that finds the record with the given sku. Inside the search, the <DNA_FoundItems> context encloses a table displaying information about the product, in this case, other descriptive data from the TeaRoom.db database.

```
01| <!--HAS_WEBDNA_TAGS-->
02| <HTML>
03| <HEAD>
04| <TITLE>Product Detail for <DNA_SKU></TITLE>
05| </HEAD>
06| <DNA_include file=TeaRoom_Header.inc>
07| <P>
08| <CENTER>
09| <DNA_search db=TeaRoom.db&eqskudata=<DNA_url><sku></DNA_
    url>>
10| <DNA_founditems>
11| <H3>Product detail for <DNA_sku> - <DNA_title></H3>
12| <TABLE BORDER=0 CELLPADDING=4 CELLSPACING=4WIDTH=500>
13| <TR>
14| <TD ALIGN=CENTER>PRODUCT # <DNA_sku></TD>
15| <TD ALIGN=CENTER VALIGN=MIDDLE><FONT SIZE=5><DNA_
    title></FONT></TD>
16| </TR>
17| <TR>
18| <TR>
19| <TD ALIGN=CENTER VALIGN=TOP ROWSPAN=8><DNA_showif<DNA_
    hasPhoto>=T><IMG ALIGN="CENTER" SRC="<DNA_showif
    Windows=<DNA_platform>>/WebCatalog/Art/</DNA_
    showif>TeaRoomImages/<DNA_PhotoName>" ></DNA_showif> <DNA_
    showif<DNA_hasPhoto>!T>No Photo Available</DNA_showif></TD>
20| <TD ALIGN=CENTER VALIGN=TOP COLSPAN=2><FONT
    SIZE=4><DNA_description2></FONT></TD>
21| </TR>
```

Most shoppers want to see the products they are buying. In line 19| we give them that opportunity. There are two fields in our database that enable us to do this. The first field is named "hasPhoto" and can have a value of "T" or "F". The second field is named "PhotoName" and its value is the name of an image file in a folder or directory called "TeaRoomImages".

Line 19] begins by defining some of the details about the table cell which are designed to enable the display of different sized pictures without misaligning the descriptive text that is also part of the table.

The `<DNA_Showif>` context defines the following condition. If the database field `hasPhoto` has a value of `T`, then display the image in the folder `TeaRoomImages` whose source file is named `PhotoName`, where the actual name of the photo as entered in the database is substituted for the fieldname tag `<DNA_PhotoName>`. It is important to note that the actual photo is not stored in the database, only its name is stored.

Conversely, if the value of the `hasPhoto` field for this record is `"F"`, then the text `"Photo Not Available"` is displayed. By using the `<DNA_Showif>` context our shopper has the ability to examine the product before buying it, even before putting it in the shopping cart.

```
21| <TR>
22| <TD ALIGN=CENTER VALIGN=TOP><FONT
    SIZE=3><DNA_Description3></FONT></TD>
23| </TR>
24| <TR>
25| <TD ALIGN=LEFT ROWSPAN=1><DNA_Description1></TD>
26| </TR>
27| <TR>
28| <TD ALIGN=LEFT><DNA_hideif<DNA_url> <DNA_Ingredients>
    </DNA_url>=>IN GREDIENTS: <DNA_Ingredients> </DNA_
    hideif></TD>
29| </TR>
30| <TR>
31| <TD ALIGN=LEFT>Price: $<DNA_format .2f> <DNA_price> </DNA_
    format></TD>
32| </TR>
33| <TR>
34| <TD ALIGN=LEFT><a
    href="ShoppingCart.tpl?command=add&db=TeaRoom.db&cart=<DNA_cart
    >&sku=<DNA_url><DNA_sku></DNA_url>">Add to Cart</a></TD>
35| </TR>
36| <TR>
37| <TD ALIGN=CENTER></TD>
38| </TR>
39| </TABLE>
40| </DNA_founditems>
41| </DNA_search>
42| Press your browser's &quot;Go Back&quot; button to see the page you just
    came from,<BR>or return to the <A
    HREF="Search.tpl?cart=<DNA_cart>">Search by Category Page</A>
```

```
43| </CENTER>
44| </BODY>
45| </HTML>
```

The remaining lines in this template should look similar to the Results.tpl page as well.

Now that you have shopped in the store, examined the products of interest and dropped them into your shopping cart, it's time for you to make your final purchase decision and for WebDNA to total up the charges and take your order.

In the simulation, you have decided to buy only the Cinnamon Plum tea. You need to tell WebDNA where the product is to be shipped so it can make the calculations required to total up the charges for your order. Accept the UPS Ground default shipping method, click on the California radio button at the bottom of the ShoppingCart.tpl page and then click on the Proceed to Final Checkout button to access the Invoice.tpl page, as shown in Figure 9. If you made the selections suggested in this tutorial, your page will look like the following one:



Figure 9. Final Checkout Page

STEP 6: USING THE PURCHASE/INVOICE PAGE

Tom and Katy's reason for developing the TeaRoom commerce site was to offer their products for sale over the World Wide Web. You've finally reached the point in the tutorial where you can see that happen.

After all, shopping is not buying!

```
01| <!--HAS_WEBDNA_TAGS-->
02| <HTML>
03| <HEAD>
04| <TITLE>Invoice</TITLE>
05| </HEAD>
06| <DNA_include file=TeaRoom_Header.inc>
07| <DNA_orderfile cart=<DNA_cart>>
08| <DNA_SetHeader
cart=<DNA_cart>>shipVia=<DNA_url><DNA__shipVia>
  </DNA_url>&shipToState=<DNA_url><DNA__shipToState>
  </DNA_url>&CartIPAddress=<DNA_ipAddress><DNA_SetHeader>
09| <DNA_lineitems>
10| <DNA_SetLineItem
  cart=<DNA_cart>&index=<DNA_lineindex>>>quantity=<DNA_interpret>
  <DNA_quantity><DNA_lineindex>></DNA_interpret>&textA=<DNA_url>
  <DNA_lookup db=TeaRoom.db&lookinfield=sku&value=<DNA_url>
  <DNA_sku> </DNA_url>&returnfield=title
> </DNA_url> </DNA_SetLineItem>
11| </DNA_lineitems>
12| <H1>Final Invoice</H1>
13| [ <A
  HREF="ShoppingCart.tpl?command=showcart&db=TeaRoom.db&cart=<DN
  A_cart>">Change Quantities, Shipping, or Ship-To State</A> |
14| <A HREF="Search.tpl?cart=<DNA_cart>">Shop Some More</A> ]<BR>
15| <H3>Your Final Invoice Contains the Following Items<P>
16| Your Order Number is #<DNA_cart></H3>
```

Line 07] uses the <DNA_OrderFile> context that allows access to all the information stored in the shopping cart file referenced by the value of the cart variable. In particular, the <DNA_orderfile> context lets you view all the shipping and line item information relating to a specific order. Since the user may have changed the quantities of items purchased, and entered some shipping information, on the shoppingcart.tpl page that linked to this page, we must first update the information in the current shopping cart.

Line 08] sets the shipping method selected in the pop-up menu on the shoppingcart page. This information is stored in the header area of the

shopping cart and is set using the <DNA_SetHeader> context. Since you are inside of the <DNA_OrderFile> context, you can see the current value set in the shipvia field, for example, by simply placing square brackets around the name: <DNA_ShipVia>.

Remember that you chose to place two underscores in front of the field names on the results page. The reason you chose to do this was twofold. First, and most important, we didn't want the value of the incoming form variable to conflict with the value currently in the shopping cart. That is, if the form variable was named shipvia, then simply placing <DNA_ShipVia> in your template would create an ambiguous situation - which shipvia do you want, the one in the shopping cart, or the one in the form? Secondly, however, we wanted the name to be similar to the name of the field in the shopping cart so we can read the template easily. The two underscores preceding the name mean that the name doesn't conflict with the shopping cart value, and yet make it obvious what the field contains.

Lines 09| to 11| update the quantities of the individual line items in the shopping cart. Since they might have changed all the quantities, you must loop through all the line items with the <DNA_LineItems> context and reset the value of the quantity variable. Quantity is a value that is stored in each individual line item and is only available within the <DNA_lineitems> context. Notice that the quantity form is named variables on the results page so they could be sequentially in the <DNA_LineItems> context using the <DNA_LineIndex> as a counter. In addition to quantity, the title of the product is placed in the TextA field using the <DNA_LookUp> tag. <DNA_LookUp> is a very specialized, and very fast, form of database search. Since the product title is not normally saved in the shopping cart, we place the title in one of the 5 spare fields - TextA - TextE.

Line 13| lets you return to the Results.tpl page so you can update the quantity or shipping information selected.

Line 16| uses the <DNA_Cart> tag to assign the cart token number to this order form. Katy and Tom will be able to use Administration templates to call up any of the orders in the Orders folder. Note that the order number is not intended to be used as an invoice number although it could be used that way. However, invoice numbers are usually numbered sequentially. WebDNA <DNA_Cart> tokens are not created in sequential order.

```
17| <FORM METHOD="POST" ACTION="ThankYou.tpl">  
18| <INPUT TYPE="HIDDEN" NAME="cart" VALUE="<DNA_cart">">
```

```

19| <TABLE WIDTH=520 BORDER="1" CELLPADDING="1">
20| <TR>
21| <TH WIDTH="38">Item #</TH>
22| <TH WIDTH="45">Qty</TH>
23| <TH WIDTH="65">Product #</TH>
24| <TH WIDTH="250">Product Name</TH>
25| <TH WIDTH="45">Price</TH>
26| <TH WIDTH="55">Extension</TH>
27| </TR>
28| <DNA_lineltems>
29| <TR>
30| <TD ALIGN="CENTER" WIDTH="38"> <DNA_lineIndex></TD>
31| <TD ALIGN="CENTER" WIDTH="45"> <DNA_quantity></TD>
32| <TD ALIGN="CENTER" WIDTH="65"> <DNA_sku></TD>
33| <TD WIDTH="250"> <DNA_textA></TD>
34| <TD ALIGN="RIGHT" WIDTH="45">$ <DNA_price></TD>
35| <TD ALIGN="RIGHT" WIDTH="65">$ <DNA_format
.f> <DNA_math>
    <DNA_quantity>*<DNA_price></DNA_math></DNA_format></TD>
36| </TR>
37| </DNA_lineltems>

```

Lines 17| and 18| open the <FORM> tag (You need more input from the buyer.) and define the template to be returned, ThankYou.tpl.

Lines 19| to 28| open the <TABLE> tag, define the column widths and table headings, and open a <DNA_Lineltems> context.

Lines 30| to 36| require some explanation. If you look at the shopping cart file format in the File Formats section of this manual you'll see that the line items part of the file includes only the following fields:

sku,quantity,price,taxable,canEmail,unitshipCost,textA,textB,textC,textD, and textE. There is no mention of fieldname tags such as <DNA_title>, <DNA_description1>, <DNA_description2>, etc.. The reason for this is that

WebDNA allows you to name fields in your database any way you like. The TextA through TextE line items act as "placeholders" for any five field name tags you may want to include in our order file. On our TeaRoom site each line item on our final purchase invoice incorporates the <DNA_Title> fieldname tag to describe what the purchaser is buying.

Lines 30| through 35| fill in the remainder of the columns in the <DNA_Lineltems> context with the values for quantity (derived from the value entered in the field on the shopping cart page), SKU, product title, price, and extended price.

Until now most of what's been done with WebDNA has been aimed at searching and displaying information from the database. The lines of HTML and WebDNA that follow implement some of the basic commerce aspects of WebDNA. For a complete discussion of WebDNA's commerce features refer to the reference section of this manual and the on-line reference.

```
38| <TR>
39| <TD ALIGN="RIGHT" COLSPAN="5"> <B>Subtotal</B></TD>
40| <TD ALIGN="RIGHT" WIDTH="65"><B>$<DNA_subTotal></B></TD>
41| </TR>
42| <TR>
43| <TD ALIGN="RIGHT" COLSPAN="5"><B><DNA_ShipToState> Tax Rate:
   <DNA_taxRate>% Tax</B></TD>
44| <TD ALIGN="RIGHT" WIDTH="65"><B>$<DNA_taxTotal></B></TD>
45| </TR>
46| <TR>
47| <TD ALIGN="RIGHT" COLSPAN="5"><B><DNA_shipVia> Shipping &
   Handling</B></TD>
48| <TD ALIGN="RIGHT" WIDTH="65"><B>$<DNA_shippingTotal></B> </TD>
49| </TR>
50| <TR>
51| <TD ALIGN="RIGHT" COLSPAN="5"><B>Grand Total</B></TD>
52| <TD ALIGN="RIGHT" WIDTH="65"><B>$<DNA_grandTotal></B></TD>
53| </TR>
54| </TABLE>
```

Lines 38| to 54| introduce new WebDNA tags dedicated exclusively to WebDNA's commerce functions.

Line 40| introduces the `<DNA_SubTotal>` tag. This tag calculates the sum of the extended prices of all the line items in the `<DNA_LineItems>` context.

Quite a few things are going on in lines 43| and 44| even though it appears to be simply another labeling for the adjacent `<DNA_TaxTotal>` tag. This line demonstrates another powerful commerce feature of WebDNA.

The value of `<DNA_ShipToState>` was determined when the shopper clicked on one of the radio buttons on the `ShoppingCart.tpl` page. The possible values were CA for California, VT for Vermont, and Other. Here the tag is used to inform the visitor which state's tax rate is being used to calculate the `<DNA_TaxTotal>`. The value of `<DNA_TaxTotal>` will be calculated by multiplying the value of `<DNA_SubTotal>` by the `<DNA_TaxRate>`. Remember, however, that there are three possible tax rates. One tax rate must be charged for shipments made to Vermont, yet another for shipments made to California, and none at all for shipments made to other destinations. How can WebDNA determine which value to substitute for the `<DNA_TaxRate>` tag?

Enter the Formulas.db database, the TaxRates.db database and the <DNA_LookUp> tag.

When a shopper clicks on one of the Add to Cart hypertext links and issues an add command to add products to their shopping cart, WebDNA gets the price of the product in one of two ways: the price can come from a field in the database called “price”, or it can be calculated based on a formula stored in a separate database called the “Formulas.db”.

Note: To prevent “hacking”, WebDNA only allows remote users to set product prices if the Price Change Password defined in the WebDNA Preferences is used. However, you can still customize pricing by creating a formula that calculates a different price based on any WebDNA tag, such as <DNA_UserName>, <DNA_Zip>, <DNA_AccountNumber> or even a calculation within a <DNA_Math> context. This TeaRoom tutorial uses formulas to calculate price, taxRate, unitShipCost and overall shipping costs based on some component of the customer’s shipping address such as state, or zip code.

Every time a product is added to the shopping cart, WebDNA calculates the item’s price, unit ship cost, ship cost, and tax rate in the following manner:

WebDNA first looks for a file called Formulas.db in the same folder as the shopping cart template itself, and looks there for a formula named “price”. If the Formulas.db database has a price formula, WebDNA evaluates the WebDNA expression comprising the formula within the context of the current shopping cart file, so that the value of tags such as <DNA_Zip> and <DNA_SKU> are available for the calculation. Then it sets the price of the product based on the calculated formula, or if no formula is found, simply uses the value of the price field corresponding to the item’s SKU from the product database. WebDNA repeats the same steps for unitShipCost as well.

For <DNA_TaxRate> and <DNA_ShipCost> WebDNA repeats the same process. However, <DNA_TaxRate> and <DNA_ShipCost> are applied to the entire order, not just the one item that was added. If no formulas for <DNA_TaxRate> or <DNA_ShipCost> are found in the Formulas.db database, WebDNA looks for form variables called “taxRate” and “shipCost” and uses them instead.

The TeaRoom uses formulas to calculate the price, tax rate and ship cost amounts. Review the Formulas.db and TaxRates.db to see how they work together to calculate the value of <DNA_TaxRate> for each purchase.

In this case the formula for taxRate takes the form of a <DNA_LookUp> tag that looks like this as a record in the Formulas.db database:

```
taxRate<tab><DNA_lookup
    db=TaxRates.db&lookInField=State&value=<DNA_ShipTostate>&returnField=taxRate&notFound=0.00><cr>
```

A <DNA_LookUp> tag performs an extremely fast search through the specified database and returns either the value of the returnField in the found record, or the literal text of the notFound value. The search is an exact match, case-sensitive, so “CA” does not equal “ca”. The syntax for a <DNA_LookUp> tag has five parts.

- taxRate<DNA_lookup db=TaxRates.db
- &lookInField=State
- &value=<DNA_ShipTostate>
- &returnField=taxRate
- ¬Found=0.00>

Examining the TaxRates.db database. will help you to understand how this works. It contains the following field names and data:

State	taxRate
CA	8.25
VT	5.0
Other	0.00

Suppose that the shopper clicked on the California radio button on the ShoppingCart.tpl template. That set the value of <DNA_ShipToState> to CA. The taxRate formula looks in the TaxRates.db database (part 1) in the State field (part 2) for a value equal to <DNA_ShipToState> (“CA”, part 3) and returns its associated taxRate (“8.25”, part 4), as shown in Figure 10. If it finds no record matching <DNA_ShipToState>. i.e. “CA”, “VT”, or “Other”, it returns the literal string assigned to notFound, in this case “0.00”.

Therefore, using the example above, the value of <DNA_taxRate> in line 43| will be 8.25. WebDNA will multiply <DNA_subTotal> by 8.25% (.0825) to calculate the value of <DNA_taxTotal> in line 44|.

The same process is used to calculate the value of <DNA_ShippingTotal> in line 48|.

Lines 51| to 52| create the Grand Total label and use the <DNA_GrandTotal> tag to calculate the sum of <DNA_SubTotal>, <DNA_TaxTotal>, and <DNA_ShippingTotal>.

```
55| Please enter the following information so that we may process your
    | order.<P>
56| <P>
57| <CENTER>
58| <DNA_include file=InvoiceInfo.inc>
59| <P>
60| <INPUT TYPE="SUBMIT" VALUE="Purchase"></FORM>
61| </CENTER>
62| </DNA_orderfile>
63| </BODY>
64| </HTML>
```

Line 58| uses an `<DNA_include>` tag to incorporate the HTML form elements needed to get input for Bill To: and Ship To: information from the shopper so we can complete the transaction. Open the file `InvoiceInfo.txt` if you want to examine the HTML and WebDNA it used to capture this information. The only part of this file that requires your special attention is the inclusion of the `<DNA_ShipTostate>` tag in the Ship To: part of the form. We want to pass this value on to the `ThankYou.tpl` template because WebDNA will use it to calculate the `taxRate` and `shipCost` values for that template as well as this one.

Line 60| defines the submit button for the form elements defined in all the previous lines.



Tea Room • Bed & Breakfast • Gift Shop
 RR5-31A Chester, VT 05143 (802)875-4767
 (At the corner of School and Canal Streets - Turn south at the Red Brick Church)

Final Invoice

[\[Change Quantities, Shipping, or Ship-To State \]](#) [\[Shop Some More \]](#)

Your Final Invoice Contains the Following Items

Your Order Number is #9580542703073

Item #	Qty	Product #	Product Name	Price	Extension
1	1	10003	Cinnamon Plum	\$10.00	\$10.00
Subtotal					\$10.00
CA Tax Rate: 8.25% Tax					\$0.83
UPS Shipping & Handling					\$6.00
Grand Total					\$16.83

Please enter the following information so that we may process your order.

Figure 10. Final Invoice Page (order portion)

Please enter the following information so that we may process your order.

Bill To	
Name:	<input type="text"/>
Address:	<input type="text"/> <input type="text"/>
City:	<input type="text"/>
State:	<input type="text"/>
Zip:	<input type="text"/>
Country:	<input type="text"/>
Phone:	<input type="text"/>
Email:	<input type="text"/>
Ship To (If different from Bill To)	
Name:	<input type="text"/>
Address:	<input type="text"/> <input type="text"/>
City:	<input type="text"/>
State:	<input type="text"/>
Zip:	<input type="text"/>
Country:	<input type="text"/>
Phone:	<input type="text"/>
Email:	<input type="text"/>
Payment	
Card	Visa
Type:	<input type="radio"/> Credit Card Purchase <input checked="" type="radio"/> Demo Account Authorizer
Expires:	January 1997
Acct #:	4111111111111111 "4111111111111111" for Demo Purchase

Figure 11. Final Invoice Page (billing, shipping, payment)

Finalize your order for Cinnamon Plum tea by clicking on the Purchase button, as shown at the bottom of Figure 11. Your browser will display the ThankYou.tpl page:

STEP 7: ACKNOWLEDGING THE ORDER

If you ordered a product in a store for later delivery, you would expect a receipt for your money or some evidence that you had paid for the products you ordered. The tutorial doesn't do anything quite so advanced, but it does acknowledge the purchase the customer made.

This page is almost exactly the same page as the Invoice.tpl page. It simply replaces the input fields with WebDNA variable and fieldname tags to display the information input by the customer. Examine the lines of HTML/WebDNA

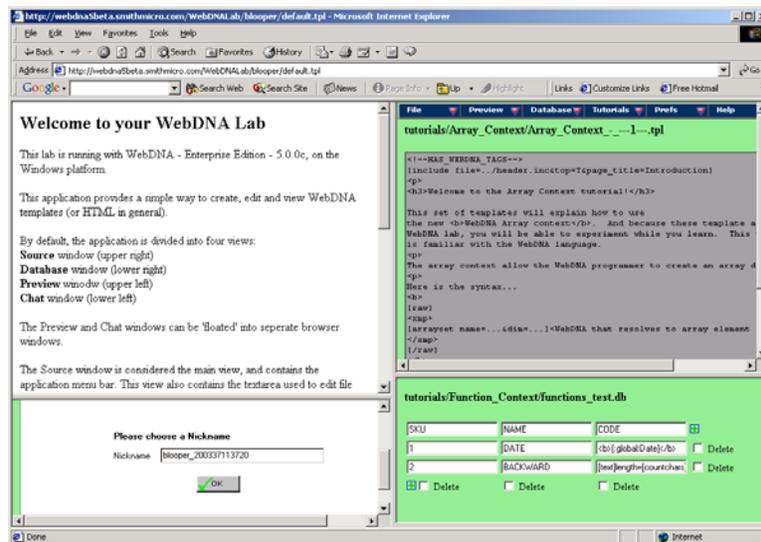
below and the contents of the ThankYouInfo.txt file that is used to send an email to the customer to see how this page works.

WebDNA Lab

This application provides a simple way to create, edit and view WebDNA templates (or HTML in general).

By default, the application is divided into four views:

- Source** window (upper right)
- Database** window (lower right)
- Preview** window (upper left)
- Chat** window (lower left)



The Preview and Chat windows can be 'floated' into separate browser windows.

The Source window is considered the main view, and contains the application menu bar. This view also contains the textarea used to edit file content.

The Database view is used to edit database files.

The Preview window is used to view 'rendered' WebDNA/HTML files.

The Chat window contains a java based chat control that enables a lab user to join in live chat sessions with other WebDNA lab users.

The menu bar contains the following...

- **Edit Menu** - Contains all the options needed to load, create, edit, and delete WebDNA/HTML files. It will also display a history of the last eight files opened in the Source window.
- **Preview Menu** - Allows you to load and refresh the contents of the 'Preview' window. The Preview window is also automatically refreshed when a file or database is saved.
- **Database Menu** - Similar to the Edit menu, the Database menu contains the options needed to create, load, delete, and edit databases.
- **Prefs Menu** - Contains misc. configuration options. Also contains a link to the Sandbox Admin templates (if this lab is designated as an ISP Sandbox).
- **Help Menu** - Contains a link to the online WebDNA guide. Also contains links to several WebDNA tutorials.

Where to Go from Here?

Additional commerce functionality is built into WebDNA. This includes the ability to send emails containing order file information to customers, acknowledging their orders, and notifying the person responsible for fulfilling received orders. Use WebDNA tags and contexts to define the text for those email messages. Additional email oriented tags are used to define the sender, recipient, and subject fields required for Internet email.

Refer to the *WebDNA 5.0 At-A-Glance Reference* in Chapter 3 and the *Advanced Uses of WebDNA* in Chapter 4 sections for details on how to set up the Email Program and its required preferences.

Chapter 3 – WebDNA Reference

WebDNA is very powerful and requires much less code than most other web programming languages. Refer to this section as needed for all available tag, context and command details at the time of this manual's printing. Refer to the WebDNA Software Corporation web site (www.webdna.us) for any updates, additions and current examples of use.

WebDNA 5.0 At-A-Glance Reference

The *WebDNA 5.0 At-A-Glance Reference* provides a list of all tags, contexts and commands used by WebDNA as of the printing of this document. They are grouped under the following categories making it easier to determine which one to select for what function:

- Searching
- Databases
- Shopping
- Showing and Hiding
- Dates and Times
- Text Manipulation
- Passwords
- Files and Folders
- Technical
- Browser Info
- Miscellaneous
- File Formats

Pay special attention to those items marked as deprecated. Though still supported in this version, these will not be supported in the next version of WebDNA.

Searching

[FOUNDITEMS] CONTEXT

Syntax: [founditems]Database Fields[/founditems]

Result: Loops through the number of items specified in the list of found records.

Required Enclosing Context:

[search]...[founditems]...[/founditems]...[/search]

Optional Context Tags:

- **[fieldname]** – Name of any field in the database being searched. The value of the field specified for a record that matched the search criteria is returned.
- **[index]** – A number from 1 to the number of matching records indicating this record's index in the list.

To display a list of all matching records from a search command or a [search] context, insert a [FoundItems] context into a WebDNA template. If the search specified a maximum number of matches, the maximum matches found are displayed in the [FoundItems] loop. Also see [ShowNext].

For example:

```
[FoundItems]
Name: [name]<br>
Address: [address]<br>
Phone: [phone]<br>
<hr>
[/FoundItems]
```

[LOOKUP] TAG

Syntax: [LookUp

db=databasePath&value=searchValue&lookInField=searchField&returnField=fieldName¬Found=TextIfNotFound]

Placing [LookUp] in your template performs an extremely fast search through the specified database and returns either the value of the returnField in the found record, or the literal text of the notFound value. The search is an case-

sensitive, exact match, so “Grant” does **not** equal “grant.” If you want more control over the search criteria, use a [Search] context instead.

[SEARCH] CONTEXT

Syntax: [search criteria&max=n]Found Items[/search]

Result: Searches a database for matching records and displays the found matches.

Optional Tag Parameter:

- **search criteria** – See Search command.
- **max=number** – The max parameter determines the maximum number of records that will be displayed in the [FoundItems] loop for the search. Additional found records will have links built for them by the [ShowNext] context.
- **startat=number** – The starting index number in the found items to display in the [FoundItems] context. The default value for this parameter is 1. The [ShowNext] context automatically adds this value so you rarely need to set it yourself.

Optional Context Tags:

- **[numfound]** – A number indicating how many records matched the search request.
- **[founditems][founditems]** – Displays the records matching the search and following the values set for startat and max.
- **[shownext][shownext]** – Displays links for the items found but not listed in the [founditems] context.
- **[Sum field=fieldName]** Calculates the numerical sum of all the found records, using the *fieldName* column.
- **[Avg field=fieldName]** Calculates the numerical average of all the found records, using the *fieldName* column.
- **[Min field=fieldName]** Calculates the numerical minimum of all the found records, using the *fieldName* column.
- **[Max field=fieldName]** Calculates the numerical maximum of all the found records, using the *fieldName* column.

For example:

```
[search db=names.db&eqNAMEdata=Grant]
Found [numfound] items<br>
[founditems]
<br>
[Name], [Address], [City]<br>
[/founditems]
[/search]
```

Whenever WebDNA encounters a [Search] context, it immediately opens the database and searches through it based on the search criteria in the search parameters. You will almost always place a [FoundItems]...[/FoundItems] context inside the [search] context in order to display the information from the matching records.

You can also substitute any [xxx] tags in the search parameters, as in [search db=[userDB]&eqNAMEdata=[username]].

Parameter	Description
db	Name of the database you wish to search
max	A number indicating how many records should be displayed at once before showing a list of "Show Items xx-yy" hyperlinks

See the *Search command* for more information on searching.

SEARCH COMMAND

Syntax: Search?db=DatabaseName&search1data=xx&search2data=yy

Searches for matching records in a database and returns the found ones.

The search command looks through every record in the database and returns records matching the search criteria. It is extremely fast and flexible - you can search through multiple fields and sort the results in just about any way imaginable. The "matched" or "hit" records in the database are displayed in a [FoundItems] loop.

SQL/ODBC Note: To search through ODBC-compliant database, use the [SQL] context.

Searching can be very sophisticated, so we have dedicated a complete section to it here.

For example, normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=Search&db=SomeDatabase.db&eqNAMEdata=Grant>

The database "SomeDatabase.db" opens, all records whose name field is "Grant" found, and any [FoundItems]..[/FoundItems] contexts in the template xx.tpl filled with the list of found records.

Here are some equivalent ways to send the same command:

HTML Source	Description
<pre></pre>	Hyperlink command
<pre><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="Search"> <input type="hidden" name="template" value="xx.tpl"> <input type="hidden" name="db" value="SomeDatabase.db"> <input name="eqNAMEdata" value="Grant"> <input type="submit"> </form></pre>	Form-based command

SEARCHING COMPARISONS

Searches using comparisons of data.

You can compare data during a search in many different ways -- the field may match exactly, greater than, less than, in a range, etc. You tell WebDNA how to compare fields in a database by typing the field name followed by **comp** into a URL. As a shortcut, you can also put the two-letter comparison code in front of the fieldname's **data** specifier.

If your database has a field called "FirstName", and you want to look for records that are alphabetically arranged after "Grant" in the FirstName field, then you would enter a URL something like this:

```
http://Results.tpl?command=search&FirstNamedata=Grant&FirstNamecomp
=gt&db=SomeDatabase.db
```

shortcut equivalent form for the search above:

```
http://Results.tpl?command=search&gtFirstNamedata=Grant&db=SomeData
base.db
```

The template "Results.tpl" displays after the search, and any [FoundItems] contexts will be filled with found records. "gt" stands for "greater than," meaning the FirstName field must be greater than "Grant," otherwise it will not be considered a match.

Note: Probably the most commonly used (and simplest) way to search a database is the "Word Or" search. This most closely matches Yahoo and Lycos' style of searching, because it finds all records matching at least one word entered by the visitor, and sorts the "best match" records to the top of the list. If you have a product database with a **Description** field, the search form might look like this:

```
<form method="POST" action="Results.tpl">
<input type=hidden name="command" value="Search">
Description: <input name="woDescriptiondata"> <input type="submit">
</form>
```

Below is a list of all the different ways to compare information in a field. All comparisons are not case-sensitive (unless you force them to be), so "JOE" is the same as "Joe":

Comp	Description
ls	less than - lsFirstName data =Grant finds all records whose FirstName field is "less than" Grant. Comparisons can be alphabetic, numeric, or date, depending on the type you specify for the field.
le	less than or equal to - leFirstName data =Grant finds all records whose FirstName field is "less than or equal to" Grant.
eq	equal to (default option for numbers) - eqFirstName data =Grant finds all records whose

	FirstName field is "equal to" Grant (upper and lower case do not matter unless you want them to: see Case Sensitivity)
ge	greater than or equal to - geFirstNamedata=Grant finds all records whose FirstName field is "greater than or equal to" Grant.
gr	greater than - grFirstNamedata=Grant finds all records whose FirstName field is "greater than" Grant.
rn	<p>range: field contains two values separated by spaces (or any other delimiter specified by wbrk). The values may be dates, numbers, or text. You must specify what type of data the fields are by using <i>fieldnametype=xxx</i> (where xxx is num or date).</p> <p>rnZipCodedata=92069 93090&ZipCodetype=num finds all records whose ZipCode field is numerically in the range of 92069 - 93090</p> <p>Note: When specifying a range, the smaller value must precede the larger value, i.e. rnZipCodedata=92069 93090, not rnZipCodedata=93090 92069.</p>
mr	minimum range value : if no maximum then ge is used instead
xr	maximum range value : if no minimum than le is used instead
ne	not equal - neFirstNamedata=Grant finds all records whose FirstName field is "not equal" Grant.
bw	begins with - bwFirstNamedata=Jo finds all records whose FirstName field "begins with" Jo, as in "Joe", "Joseph", "Josephine".
cl	<p>close to (numeric only).</p> <p>clZipCodedata=92069&clZipCodedata=10 finds all records whose ZipCode field is within 10 of 92069 (92059 - 92079)</p>
ws	Interpret the words as a single string to be matched (including spaces, etc.) This lets you find entire phrases

	(including spaces etc.) This lets you find entire phrases, like "Joe enjoys butter" only if those 3 words are next to each other in that order, including spaces (unlike wa and wo , below)
wa	Separate the w ords and " and " them together (all must match). Searching for 3 words using wa will match only if all 3 are in the field, but not necessarily next to each other.
wo	Separate the w ords and " or " them together (at least one must match - default option for text). Search for 3 words using wo will match if any one of them matches..
wn	w ord n ot equal - none of the words match text in the specified field

[SHOWNEXT] CONTEXT

Syntax: [shownext position=value&method=value&max=number]form or
hypertext link[/shownext]

Result: Creates a list of hypertext links or buttons that display more found items.

Optional Context Parameters:

- **position=value** – Acceptable values are Begin, Middle, or End. Begin loops through “chunks” of indices preceding the ones displayed inside the [foundItems] loop. Middle displays the range of indices that are inside [foundItems] loop (rarely, if ever, used). End loops through “chunks” of indices after the ones displayed inside the [foundItems] loop. If no position is specified, the default is to loop through both beginning and end “chunks”, thus displaying all possible [Show Next] hypertext links.
- **method=value** – Acceptable values are GET - [SearchString] will display HREF-style (hypertext link) parameters for the search, and POST - [SearchString] will display FORM METHOD=POST style parameters for the search. Use this only if your search is so complex that it requires more than 255 characters worth of search parameters.

If no method is specified, then GET is assumed, which means that HREF-style links are generated.

To display a list of hypertext links that will display more items found in a search, insert one or more [ShowNext] contexts inside a [Search] context. This context is only valid inside a [search] context, and will only be used if the search specified a max=N, and the number of found items is greater than max.

Optional Context Tags:

- **[start]** – A number indicating the index of the first item that will be displayed in this range of found items.
- **[end]** – A number indicating the index of the last item that will be displayed in this range of found items.
- **[searchstring]** – All search parameters necessary to find this “chunk” of items in the range [Start] - [End]. Either a URL-style string for method GET or HTML input tags for method POST.

For example:

```
[search search string]
[shownext position=begin]
<a href="SearchResults.tpl?command=search&[searchstring]">
Show Items
[start]-[end]</a><br>
[/shownext]
[foundItems]
[/foundItems]
[shownext position=end]
<a href="SearchResults.tpl?command=search&[searchstring]">
Show Items
[start]-[end]</a><br>
[/shownext]
[/search]
```

If a search returns 50 found items, and the max is set to 10, then [ShowNext] loops through all the “chunks” of 10 necessary to create hypertext links for each set of 10 items not displayed inside the [FoundItems] loop:

```
Show Items 1-10 (this comes from position=beginning)
Show Items 11-20
(Items 21-30 are displayed) (this comes from [foundItems])
Show Items 31-40 (this comes from position=end)
Show Items 41-50
```

If the [ShowNext] . . . [/ShowNext] context employs the GET method within an <A HREF> hypertext link, then the entire search string must be written after the ? following the search command. In other words, the entire search string, the command and search parameters, must be enclosed by the <A HREF>

If the [ShowNext] . . . [/ShowNext] context employs the POST method because the search requires parameters exceeding 255 characters, then all of the search requirements must be written within <FORM> . . . </FORM> tags and must include all the required <FORM> elements such as hidden fields which define field names in the database, field values, and a submit button.

[SQL] CONTEXT

Syntax: [sql dsn=ODBC data source&statement=sql text]founditems[/sql]

Result: Performs a SQL statement on an ODBC data source.

Required Tag Parameters:

- **dsn=ODBC data source** – Name of the database you wish to search
- **statement=text** – Any legal SQL statement. Consult an SQL reference manual for more information.
- **max=number** – A number indicating how many records should be displayed at once before showing a list of "Show Items xx-yy" hyperlinks.

Optional Tag Parameters:

- **username=text** – Username required to access this ODBC (Optional for Windows platform, if none specified, "sa" is used).
- **password=text** – Password required to access this ODBC database (Optional for Windows platform, if none is specified, a blank password is used).

Optional Context Parameters

- **[NumFound]** – A number indicating how many records matched the search request. Some ODBC drivers do not support this feature, so WebDNA compensates by visiting every record in the database in

order to count them. For large datasets, this can be very slow, and you should consider writing a SQL statement that performs a count instead. Do not put [NumFound] inside an SQL context that inserts new records because the statement will be executed twice in order to perform the count. This will cause an invalid second record to be added to your database.

- **[FoundItems]...[/FoundItems]** – Normally a [FoundItems] loop is placed inside an [SQL] context that has performed a SELECT statement. This is done so that you can display all the matching records. You can insert any database field names inside the [FoundItems] loop to display them in the HTML.

To search through an ODBC-compliant database (or add new records, or delete or replace records,) place a [SQL] context into a WebDNA template. You may specify any DSN (Data Source Name) that has been properly configured through the ODBC setup control panel on the web server computer.

The [SQL] context is not limited to searching—you may perform any legal SQL statement, such as SELECT, INSERT, DROP, etc. The SQL language is too broad to describe here; it is assumed you have a working knowledge of SQL before using this context.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[SQL dsn=Pubs&statement=SELECT * from Authors]
Found [NumFound] items<br>
[FoundItems]
[au_lname], [au_fname], [title]<br>
[/FoundItems]
[/SQL]
```

Whenever WebDNA encounters an [SQL] context, it uses ODBC to attempt to make a connection to the specified DSN. It then executes the SQL statement and retrieves the results, if any. For SQL SELECT statements, you almost always place a [FoundItems]...[/FoundItems] context inside the [SQL] context so you can display the information from the matching records.

Notice: you can substitute any [xxx] tags in the SQL parameters, as in [SQL dsn=Pubs&statement=SELECT * from Authors where au_lname > '[form.name]']].

Databases

[CLOSEDATABASE] TAG

Syntax: [CloseDatabase *db=FileName*]

Placing [CloseDatabase *db=FileName*] in your template causes the specified database file to be written and closed. This is only needed for special cases (usually before appending to a file) where you need to change a file perhaps cached in RAM. WebDNA automatically closes databases when it needs more memory, so you typically do not need to use this tag.

[COMMITDATABASE] TAG

Syntax: [CommitDatabase *db=FileName*]

Placing [CommitDatabase *db=FileName*] in your template causes the specified database file to be written but not closed (so it will remain in RAM). This is only needed for special cases where you want to be absolutely certain that a database has been written to disk.

[APPEND] CONTEXT

Syntax: [Append *db=DatabaseName*]values[/Append]

Appends a new record with the specified field values to the end of a database.

SQL/ODBC Note: To append new records to the end of an ODBC-compliant table, use the [SQL] context.

Optional Context Parameters

- **autonumber** – instructs WebDNA to automatically generate the 'next highest number' value for the given fieldname.

To add new records to a database, insert an Append context into a template. (You can also use the Append command from a URL or a FORM.) Whenever WebDNA encounters an Append context, it immediately adds a new record to the end of the specified database given the named field values inside the Append context.

For Example: Usually, the following text is inserted into a .tpl file on your server then uses a web browser to link to it:

```
[Append db=SomeDatabase.db]name=Grant&address=1492 Somewhere  
Lane&zip=90000&date=[date][Append]
```

The database “SomeDatabase.db” is opened, and a new record is added to the end. The field name “name” is set to “Grant,” the field name “address” is set to “1492 Somewhere Lane,” the field name “zip” is set to “90000,” and the field name “date” is set to the current date. Notice that any WebDNA [xxx] tags inside the context are first substituted for their real values before being written to the database. The name of the database itself may also be an [xxx] tag, as in “[Append [FormVariable]].”

Any field names not existing in the database are ignored, and any fields not specified are left blank in the new record. Certain letters are illegal, such as <tab> or <carriage return>, so they are converted to <space> and <soft return> before being added to the database.

Note: Some computers use the two-character sequence <carriage return><line feed> to indicate a single end of line, which WebDNA automatically converts to a single <soft return> character before adding to the database.

You may specify an absolute or partial path to the database file, as in “/WebDNA/Folder/SomeDatabase.db” or “../SomeDatabase.db” (relative paths start in the same folder as the template, just like URLs, so “..” will look “up” one folder level from the template, and “/” will start at the web server’s root).

Note: Some database filenames are reserved. You may not create your own database named “WebCatalog Prefs,” “Users.db,” “ErrorMessages.db,” “StandardConversions.db,” or “Triggers.db.”

As a rule, database file paths are relative to the local template, or if they begin with “/” they are relative to the web server’s virtual host root. You may optionally insert an “^” in front of the file path to indicate the file can be found in a global root folder called “Globals” inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

You can use the '**AUTONUMBER=**' parameter with the [append] or [replace] context to instruct WebDNA to automatically generate the 'next highest number' value for the given fieldname. This is useful for 'ID' type fields, where unique values are required.

Here is a demonstration of the AUTONUMBER feature using a WebDNA TABLE (of course this will work on database files as well).

Example WebDNA code:

```
[table name=table_1&fields=ID,NAME,EMAIL] [/table]

[append
  table=table_1&AUTONUMBER=ID]NAME=Scott&EMAIL=scott@here.
  com[/append]
[append
  table=table_1&AUTONUMBER=ID]NAME=Lee&EMAIL=lee@there.com
  [/append]
[append
  table=table_1&AUTONUMBER=ID]NAME=OMNI &EMAIL=omni@everywh
  ere.com[/append]
[delete table=table_1&eqIDdata=2]
[append
  table=table_1&AUTONUMBER=ID]NAME=Lee&EMAIL=lee@there.com
  [/append]

[search table=table_1&neIDdata=[blank]]
[founditems]
[ID] - [NAME] - [EMAIL]

[/founditems]
[/search]
```

Results:

```
1 - Scott - scott@here.com
3 - OMNI - omni@everywhere.com
4 - Lee - lee@there.com
```

You can see that WebDNA automatically generated the ID value by calculating the 'next largest value', given the existing ID values in the table.

[ADDFIELDS] CONTEXT

New in 5.0

Syntax: [AddFields db=...]...WebDNA...[/AddFields]

Result: The [AddFields] context adds new fields to an existing WebDNA database.

First, lets use the following code to create a new test database and run a simple search on the new database.

```
[closedatabase db=addfields_test.db]

[writefile file=addfields_test.db]ID,NAME
"1","Scott"
"2","Rusty"
"3","David"
"4","Daniel"
"5","Dustin"
[/writefile]

[search db=addfields_test.db&neIDdata=[blank]&rank=off]
[founditems]
[ID] - [NAME]

[/founditems]
[/search]
```

Results:

```
1 - Scott
2 - Rusty
3 - David
4 - Daniel
5 - Dustin
```

Now, lets use the [AddFields] context to add EMAIL and PHONE fields to the 'addfields_test.db' database, initializing the new field value with some arbitrary data. We will again perform a simple search on the same database to confirm that the new fields, and field data, have been added.

We use the following WebDNA code:

```
[AddFields db=addfields_test.db]EMAIL=us.here.com&PHONE=123-1235[/AddFields]

[search db=addfields_test.db&neIDdata=[blank]&rank=off]
[founditems]
[ID] - [NAME] - [EMAIL] - [PHONE]

[/founditems]
[/search]
```

Results:

```
1 - Scott - us.here.com - 123-1235
2 - Rusty - us.here.com - 123-1235
3 - David - us.here.com - 123-1235
4 - Daniel - us.here.com - 123-1235
5 - Dustin - us.here.com - 123-1235
```

Note that the [AddFields] context will not add fieldnames that already exist in the target database.

APPEND COMMAND

Syntax: Append?db=DatabaseName&field1=xx&field2=xx

Result: Appends a new record with the specified field values to the end of a database.

To add new records to a database, use a web browser to link to a URL containing the Append command (alternately, you may embed an [Append] context into a template). Whenever WebDNA receives an Append command, it immediately adds a new record to the end of the specified database given the named field values.

SQL/ODBC Note: To append new records to the end of an ODBC-compliant table, use the [SQL] context.

For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=Append&db=SomeDatabase.db&name=Grant&address=1492%20Somewhere%20Lane&zip=90000>

The database "SomeDatabase.db" opens, and a new record is added to the end. The field name "name" is set to "Grant," the field name "address" is set to "1492 Somewhere Lane," and the field name "zip" is set to "90000." The page sent back to the browser will be xx.tpl.

Any field names not existing in the database are ignored, and any fields you do not specify are left blank in the new record. Certain letters are illegal, such as <tab> or <carriage return>, so they are converted to <space> and <soft return> before being added to the database. Some computers use the two-character sequence <carriage return><line feed> to indicate a single end of line, which is automatically converted to a single <soft return> character before being added to the database.

You may specify a relative or full URL to the database file, as in “/WebDNA/Folder/SomeDatabase.db” or “../SomeDatabase.db.” The path to the database is always relative to the template URL, so if the database is in the same folder as the template, the path would be “SomeDatabase.db”, and if the database was one folder up, then the path would be “../SomeDatabase.db.”

Note: As a general rule, all database file paths are relative to the local template, or if they begin with “/” they are relative to the web server’s virtual host root. As of version 3.0, you may optionally put “^” in front of the file path to indicate the file can be found in a global root folder called “Globals” inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

Note: You may force a visitor to enter something into a field by using the RequiredFields parameter in the URL. Setting RequiredFields=field1+field2+field3 displays an error message if the visitor forgets to enter text into any of those three fields. RequiredFields works for all commands, not just this one.

Other ways to send the same command:

HTML Source	Description
<pre data-bbox="480 814 974 890"></pre>	Hyperlink command
<pre data-bbox="480 907 966 1134"><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="Append"> <input type="hidden" name="db" value="SomeDatabase.db"> <input name="name"> <input name="address"> <input type="submit"> </form></pre>	Form-based command

DELETE COMMAND

Syntax: Delete?db=DatabaseName&searchData=xx

Result: Deletes all matching records from the database.

To delete records from a database, use a web browser to link to a URL containing the Delete command (alternately, you may embed a [Delete] tag into a template).

Whenever WebDNA receives a Delete command, it immediately searches for all matching records and deletes them from the database.

SQL/ODBC Note: To append new records to the end of an ODBC-compliant table, use the [SQL] context.

For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=Delete&db=SomeDatabase.db&eqNAMEdata=Grant>

The database "SomeDatabase.db" opens, and all records whose name field is "Grant" are deleted. The page sent back to the browser is xx.tpl

Other ways to send the same command include:

HTML Source	Description
<code></code>	Hyperlink command
<code><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="Delete"> <<input name="eqNAMEdata"> <input type="submit"> </form></code>	Form-based command

Note: As a rule, all database file paths are relative to the local template, or if they begin with "/" they are relative to the web server's virtual host root. You may optionally put "^" in front of the file path to indicate the file can be found in a global root folder called "Globals" inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

[APPENDFILE] CONTEXT

Syntax: [AppendFile FileName]Text[/AppendFile]

Result: Writes text to the end of an existing file.

To add text to the end of an arbitrary text file, place an [AppendFile] context into a template. [AppendFile] creates a new file if one does not already exist.

All text is placed at the end of the file. The file must not be a database file currently open and in use by WebDNA. See [WriteFile] for further details.

Note: [AppendFile] does not ‘understand’ databases. To append a new record to the end of a database, use Append instead.

For Example: Usually, the following text is placed into a .tpl file on your server then uses a web browser to link to it):

```
[AppendFile SomeTextFile]Hello, my name is Grant. The time is [time]
This is a second line[/AppendFile]
```

The text file “SomeTextFile” opens, and displays the following text:

```
Hello, my name is Grant. The time is 13:43:01
```

A second line is written at the end of the file. Notice that carriage returns inside the context are written to the file exactly as they appear. Also notice that any WebDNA [xxx] tags inside the context are substituted for their real values before being written to the file. You may specify a full or partial path to the file, as in “/Some Folder/file.txt” (starting from the web server’s root) or “LocalFolder/file.txt” (starting in the same folder as the template file, looking down into a folder called “LocalFolder”).

Security Note: By default, all files created by WebDNA are tagged with a special code telling WebSTAR not to display them via URL. If you want files to be visible to outside browsers, use the optional settings below.

Parameter	Description
Secure	“T” for files that should be secure—WebSTAR will not display them. “F” for files that should be visible via URL—WebSTAR will display them. For example: [WriteFile secure=F&file=SomeFile]...[/WriteFile]
File	When you use the secure option above, you must also provide the name (or relative path) of the file to create.

[EXCLUSIVELOCK] CONTEXT

Syntax: [ExclusiveLock *database list*]...WebDNA...[/ExclusiveLock]

Result: Prevents other threads from simultaneously accessing a group of databases.

SQL/ODBC Note: all database, table, and record locking mechanisms are entirely controlled by the SQL server. [ExclusiveLock] is used only for WebDNA-native databases.

To prevent a group of databases from being modified by other threads (other 'hits' to the server, or other templates or triggers), wrap an [ExclusiveLock] context around the WebDNA code that will be making the important exclusive changes.

For Example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[ExclusiveLock db=orders.db&db=lineitems.db&db=accesslog.db]
...search, delete, or modify any of orders.db, lineitems.db, or
accesslog.db while being assured that no other threads can modify any
of these databases until the closing /ExclusiveLock tag is reached.
[/ExclusiveLock]
```

The list of database names is first alphabetized to maintain a consistent locking order (a technique which prevents internal deadlocks). Then, each database lock is acquired, one at a time, until all locks are acquired. Then, the interior WebDNA is executed. If any lock cannot be acquired, the other databases are unlocked and the interior WebDNA is not executed.

Parameter	Description
db	path to first database file, relative to this template.
db	path to second database file, relative to this template.
db	...path to nth database file, relative to this template. Note that all parameters are named exactly the same: "db"

[DELETE] TAG

Syntax: [Delete db=DatabasePath&eqNAMEdata=Fred]

Placing [Delete db=DatabasePath&eqNAMEdata=Fred] in your template opens the database specified by DatabasePath, finds all the records whose Name field contains "Fred," and deletes those records. Any valid search parameters are allowed, as defined in the Search command and [Search] context.

Note: if the database has username and password fields, then the records will not be deleted unless the visitor's web browser username/password match the record's username/password.

SQL/ODBC Note: To delete records from an ODBC-compliant table, use the [SQL] context.

[FLUSHDATABASES] TAG

Syntax: [FlushDatabases]

Placing [FlushDatabases] in your template causes all databases to be written and closed. This is only needed for special cases (usually before appending to a file) where you need to change a file that may be cached in RAM, and you do not know the exact name of the database. WebDNA automatically closes databases when it needs more memory, so you typically do not need to use this tag.

[LISTDATABASES] CONTEXT

Syntax: [listdatabases]Database Tags[/listdatabases]

Result: Lists all the currently open databases.

Optional Context Tags:

- **[Name]** – The name of the database (can be a partial or full path).
- **[Index]** – A number from 1 to the number of databases, indicating this database's index in the list
- **[NumRecords]** – The number of records (rows) in this database.
- **[NumFields]** – The number of fields (columns) in this database
- **[xxx]** – Any other WebDNA tag or context, including [ListFields].

To display a list of all the databases that WebDNA currently has open, insert a [ListDatabases] context into a template. Additionally, you can insert a

[ListFields] context inside the [ListDatabases] context to display the names of all the fields in the database.

For example:

```
[listdatabases]
Name: [name]<br>
# Records: [numRecords]<br><hr>
[/listdatabases]
```

This is a very useful debugging tool. Be aware that it lists all open databases, including preference files, tax and shipping databases, user databases, etc. In other words, you may get more than you expect when you use this context.

[LISTFIELDS] CONTEXT

Syntax: [listfields databaseName]Database Tags[/listfields]

Result: Lists all the fields in the specified database.

Optional Context Tags:

- **[fieldname]** – The name of the field.
- **[index]** – A number from 1 to the number of fields, indicating this field's index in the list.

To display a list of all the fields in a particular database, insert the [ListFields] context into a template. This context may be placed inside the [ListDatabases] context to automatically list all the fields in all the databases.

For example:

```
[listdatabases]
Fields in database [name]:<br>
[listfields [name]]
Fieldname: [name]<br>
[/listfields]
<hr>
[/listdatabases]
```

[LOOKUP] TAG

Syntax: [LookUp
db=databasePath&**value**=searchValue&**lookInField**=searchField&**returnField**=fieldName&**notFound**=TextIfNotFound]

Placing [LookUp] in your template performs an extremely fast search through the specified database and returns either the value of the returnField in the found record, or the literal text of the notFound value. The search is a case-sensitive, exact match, so “Grant” does **not** equal “grant.” If you want more control over the search criteria, use a [Search] context instead.

[TABLE] CONTEXT

New in 5.0

Syntax: [table name=...&fields=,...]<fieldname 1>,<fieldname 2>,...,<fieldname n>[/table]

Result: Enables the WebDNA programmer to quickly create a temporary 'in line' database table that is local to the template and not part of the global database cache. A table can be used in any context that accepts a database 'db' as a parameter.

Optional Tag Parameters:

- **name** - A user assigned name used to reference the table during the duration of the template.
- **fields** - A comma delimited, ordered, list of fieldnames to be used for the table.

Search a Table

Lets create a table and perform a basic search on it.

We use the following code...

```
[table name=products&fields=SKU,NAME,DESC]
1001      Red Widget   A small red widget
1002      Blue Widget  A small blue widget
1003      Green Widget A small green widget
[/table]

[search table=products&neSKUdata=[blank]]
Found [numfound] items in the products table.

[founditems]
[SKU] - [DESC]

[/founditems]
[/search]
```

Results...

```
Found 3 items in the products table.  
1001 - A small red widget  
1002 - A small blue widget  
1003 - A small green widget
```

A table can be very useful if you want to perform a secondary search on the results of a previous search.

Examples with ConvertChars/Words

You can use a table to quick create a table to be use with the ConvertChars, and ConvertWords contexts

```
[table name=t1&fields=from,to]  
a  A  
b  B  
c  C  
[/table]
```

```
[table name=t2&fields=from,to]  
webdna    WebDNA  
tables     Tables  
smsi      SMSI  
[/table]
```

```
[convertchars table=t1]abc[/convertchars]
```

```
[convertwords table=t2]webdna tables, brought to you by  
smsi![/convertwords]
```

Results...

```
ABC  
WebDNA Tables, brought to you by SMSI!
```

Example with ListFiles

Tables are also very useful for sorting/searching the results of other 'iterative' WebDNA contexts. For example, lets sort the results of a [listfiles] context...

```
[table name=filesort&fields=filename,size,date][!]  
[/!][listfiles path=.]
```

```
[filename][size] [createdate]
[/listfiles][table]
```

Directory listing - sorted by filename

```
[search
  table=filesort&neFILENAMEdata=[blank]&asFILENAMEsort=1]
[founditems]
[filename] - [size] - [date]

[/founditems]
[/search]
```

Directory listing - sorted by file size

```
[search
  table=filesort&neFILENAMEdata=[blank]&asSIZEsort=1&SIZEt
  ype=num]
[founditems]
[filename] - [size] - [date]

[/founditems]
[/search]
```

Results...

Directory listing - sorted by filename

```
Table_Context_-_---1---.tpl - 1093 - 01/02/2003
Table_Context_-_---2---.tpl - 1066 - 01/02/2003
Table_Context_-_---3---.tpl - 940 - 01/02/2003
Table_Context_-_---4---.tpl - 3445 - 01/02/2003
Table_Context_-_---5---.tpl - 529 - 01/02/2003
Table_Context_-_---6---.tpl - 166 - 01/02/2003
table_tutorial5a.inc - 2255 - 01/02/2003
```

Directory listing - sorted by file size

```
Table_Context_-_---6---.tpl - 166 - 01/02/2003
Table_Context_-_---5---.tpl - 529 - 01/02/2003
Table_Context_-_---3---.tpl - 940 - 01/02/2003
Table_Context_-_---2---.tpl - 1066 - 01/02/2003
Table_Context_-_---1---.tpl - 1093 - 01/02/2003
table_tutorial5a.inc - 2255 - 01/02/2003
Table_Context_-_---4---.tpl - 3445 - 01/02/2003
```

Note that we had to comment out some line endings. This is because the WebDNA that is parsed between the [table] tags must evaluate to a list of 'records', with the 'fields' delimited by tabs and each 'record' ending with a

line ending (<cr><lf> or <lf> or <cr>). In other words, exactly the same format as a WebDNA database.

Lets combine this with a webdna [function] context to list all files in a folder, and its sub-folders.

The new WebDNA [function] context is described in the 'Function Context' tutorial.

Results...

Display the folder tree of the 'user_files' folder.

```
tutorials
  Array_Context
  Function_Context
  ISP_Sandbox
  Scope
  Special_Scripts
  Table_Context
  XML_Contexts
  XSL-XSLT_Contexts
```

Display the complete file tree of the 'user_files' folder

```
database1.db
example1.tpl
sandbox_admin_redirect.tpl
tutorials
  Array_Context
    Array_Context_-_---1---.tpl
    Array_Context_-_---2---.tpl
    Array_Context_-_---3---.tpl
    Array_Context_-_---4---.tpl
    Array_Context_-_---5---.tpl
    Array_Context_-_---6---.tpl
    Array_Context_-_---7---.tpl
    array_tutorial6a.inc
  create_tutorial.tpl
  Function_Context
    functions.db
    functions_test.db
    Function_Context_-_---1---.tpl
    Function_Context_-_---2---.tpl
    Function_Context_-_---3---.tpl
    Function_Context_-_---4---.tpl
    Function_Context_-_---5---.tpl
    Function_Context_-_---6---.tpl
    Function_Context_-_---7---.tpl
    Function_Context_-_---8---.tpl
    Function_Context_-_---9---.tpl
    function_tutorial8a.inc
  header.inc
```

```
ISP_Sandbox
  ISP_Sandbox_ _ _ _ 1 _ _ _ .tpl
  ISP_Sandbox_ _ _ _ 2 _ _ _ .tpl
  ISP_Sandbox_ _ _ _ 3 _ _ _ .tpl
  ISP_Sandbox_ _ _ _ 4 _ _ _ .tpl
  ISP_Sandbox_ _ _ _ 5 _ _ _ .tpl
  ISP_Sandbox_ _ _ _ 6 _ _ _ .tpl
Scope
  products.db
  Scope_ _ _ _ 1 _ _ _ .tpl
  Scope_ _ _ _ 2 _ _ _ .tpl
  Scope_ _ _ _ 3 _ _ _ .tpl
  Scope_ _ _ _ 4 _ _ _ .tpl
  Scope_ _ _ _ 5 _ _ _ .tpl
  Scope_ _ _ _ 6 _ _ _ .tpl
  Scope_ _ _ _ 7 _ _ _ .tpl
  Scope_ _ _ _ 8 _ _ _ .tpl
  Scope_ _ _ _ 9 _ _ _ .tpl
  Scope_tutorial7a.inc
  testcart
Special_Scripts
  Special_Scripts_ _ _ _ 1 _ _ _ .tpl
  Special_Scripts_ _ _ _ 2 _ _ _ .tpl
  Special_Scripts_ _ _ _ 3 _ _ _ .tpl
  Special_Scripts_ _ _ _ 4 _ _ _ .tpl
Table_Context
  Table_Context_ _ _ _ 1 _ _ _ .tpl
  Table_Context_ _ _ _ 2 _ _ _ .tpl
  Table_Context_ _ _ _ 3 _ _ _ .tpl
  Table_Context_ _ _ _ 4 _ _ _ .tpl
  Table_Context_ _ _ _ 5 _ _ _ .tpl
  Table_Context_ _ _ _ 6 _ _ _ .tpl
  table_tutorial5a.inc
XML_Contexts
  example1.xml
  XML_Contexts_ _ _ _ 1 _ _ _ .tpl
  XML_Contexts_ _ _ _ 10 _ _ _ .tpl
  XML_Contexts_ _ _ _ 11 _ _ _ .tpl
  XML_Contexts_ _ _ _ 12 _ _ _ .tpl
  XML_Contexts_ _ _ _ 13 _ _ _ .tpl
  XML_Contexts_ _ _ _ 14 _ _ _ .tpl
  XML_Contexts_ _ _ _ 15 _ _ _ .tpl
  XML_Contexts_ _ _ _ 2 _ _ _ .tpl
  XML_Contexts_ _ _ _ 3 _ _ _ .tpl
  XML_Contexts_ _ _ _ 4 _ _ _ .tpl
  XML_Contexts_ _ _ _ 5 _ _ _ .tpl
  XML_Contexts_ _ _ _ 6 _ _ _ .tpl
  XML_Contexts_ _ _ _ 7 _ _ _ .tpl
  XML_Contexts_ _ _ _ 8 _ _ _ .tpl
```

```

XML_Contexts_-_---9---.tpl
xml_tutorial112a.inc
xml_tutorial8a.inc
XSL-XSLT_Contexts
example1.xml
example1.xsl
example2.xsl
music.db
XSL-XSLTtutorial11.inc
XSL-XSLT_Contexts_-_---1---.tpl
XSL-XSLT_Contexts_-_---2---.tpl
XSL-XSLT_Contexts_-_---3---.tpl
XSL-XSLT_Contexts_-_---4---.tpl
XSL-XSLT_Contexts_-_---5---.tpl
XSL-XSLT_Contexts_-_---6---.tpl
XSL-XSLT_Contexts_-_---7---.tpl
XSL-XSLT_Contexts_-_---8---.tpl
XSL-XSLT_Contexts_-_---9---.tpl
welcome.tpl

```

In the example above, we used a single table to store all the folder and file info. We could have created a separate table for each folder, which would provide more flexibility for sorting the folders and files (which is what we did to create the 'file tree' popup window used in this lab application.).

[QUIT] COMMAND

Syntax: Quit

Result: Commits all databases to disk and quits the CGI.

Note: This command is no longer supported as of Version 4.0.

To commit and close all databases at once, send a Quit command to WebDNA. This is most often used when you have copied (or FTP-ed) new database files to the disk and you want WebDNA to reload them all. Plug-ins cannot quit, so if you are using the plug-in version of WebDNA, it will just commit and close databases.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

<http://www.yourserver.com/WebDNA/xx.tpl?command=Quit>

There are no parameters to the Quit command.

HTML Source	Description
<code></code>	Hyperlink to WebDNA plug-in. The template is ignored.
<code><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="Quit"> <input type="submit"> </form></code>	Form-based command to plug-in (notice the template is part of the action, but it is ignored)

[REPLACE] CONTEXT

Syntax: [replace db=databasepath&search criteria]new values[/replace]

Result: Replaces each found record in a database with the new field values.

Required Tag Parameters:

- **db=databasepath** – URL-style path to database file.
- **search criteria** – Search information that describes which records should be found and replaced. All found records are replaced with the same values. Can be any complex search criteria; works exactly like Search command or [Search] context.

Optional Tag Parameters:

- **append=Boolean** – “T” if you want a new record to be added to the end of the database in the case where no records were found to be replaced. Any fields you do not specify are left blank in the new record.
- **autonumber** – instructs WebDNA to automatically generate the 'next highest number' value for the given fieldname.

To replace records in a database, add a [Replace] context to the template (alternately, you may use the replace command from a URL or a form). Whenever WebDNA encounters a [Replace] context, it immediately searches for the specified records in the database, and replaces those records' fields with the named field values inside the Replace context.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[replace db=SomeDatabase.db&eqNAMEdata=Grant]
name=John[/replace]
```

The database SomeDatabase.db is opened, all records whose name field is Grant are found, and then set to John. All other fields in the records are left untouched. Notice that any WebDNA [xxx] tags inside the context are first substituted for their real values before being written to the database. The name of the database itself may also be an [xxx] tag, as in "[Replace db=[FormVariable]]".

Because this context replaces all found records with new values, it is useful for conducting either bulk or individual database updates. By using the append=T parameter, it is also possible to create a single template for updating databases instead of creating one template for additions, one for replacements, etc.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[Replace db=SomeDatabase.db&eqNAMEdata=Grant]
name=John&address=1492 Somewhere
Lane&zip=90000&date=[date]/[Replace]
```

The database "SomeDatabase.db" opens and all records whose **name** field is "Grant" are found. The field names "**name**" is set to "John," "**address**" is set to "1492 Somewhere Lane," "**zip**" is set to "90000," and "**date**" is set to the current date. Notice that any WebDNA [xxx] tags inside the context are first substituted for their real values before being written to the database. The name of the database itself may also be an [xxx] tag, as in "[Replace db=[FormVariable]]".

Any field names not existing in the database are ignored, and if you leave some existing field names out of the replace context, they will remain unchanged in the database. Certain letters are illegal, such as <tab> or <carriage return>, so they are converted to <soft tab> and <soft return> before being added to the database. Some computers use the two-character sequence <carriage return><line feed> to indicate a single end of line, which is automatically converted to a single <soft return> character before being added to the database. These 'soft' characters are automatically converted back to 'hard' versions (the originals) whenever you retrieve fields from a search of the database.

You may specify an absolute or relative path to the database file, as in "/WebDNA/GeneralStore/SomeDatabase.db" or "../SomeDatabase.db".

Note: Normally all database filepaths are relative to the local template, or if they begin with "/" they are relative to the web server's virtual host root (MacOS and Unix versions only; PC versions treat the DBServer.exe folder as root regardless of the virtual host). You may optionally put "^" in front of the file path to indicate the file can be found in a global root folder called "Globals" inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

You can use the '**AUTONUMBER=**' parameter with the [append] or [replace] context to instruct WebDNA to automatically generate the 'next highest number' value for the given fieldname. This is useful for 'ID' type fields, where unique values are required.

Here is a demonstration of the AUTONUMBER feature using a WebDNA TABLE (of course this will work on database files as well).

Example WebDNA code:

```
[table name=table_1&fields=ID,NAME,EMAIL][table]

[append
  table=table_1&AUTONUMBER=ID]NAME=Scott&EMAIL=scott@here.
  com[/append]
[append
  table=table_1&AUTONUMBER=ID]NAME=Lee&EMAIL=lee@there.com
  [/append]
[append
  table=table_1&AUTONUMBER=ID]NAME=OMNI &EMAIL=omni@everywh
  ere.com[/append]
[delete table=table_1&eqIDdata=2]
[append
  table=table_1&AUTONUMBER=ID]NAME=Lee&EMAIL=lee@there.com
  [/append]

[search table=table_1&neIDdata=[blank]]
[founditems]
[ID] - [NAME] - [EMAIL]

[/founditems]
[/search]
```

Results:

```
1 - Scott - scott@here.com
```

3 - OMNI - omni@everywhere.com
4 - Lee - lee@there.com

You can see that WebDNA automatically generated the ID value by calculating the 'next largest value', given the existing ID values in the table.

REPLACE COMMAND

Syntax: Replace?db=DatabaseName&searchdata=xx&field1= xx&field2=xx

Result: Replaces each found record in a database with the new field values.

To replace records in a database, use a web browser to link to a URL containing the Replace command (alternately, you may embed a [Replace] context into a template). Whenever WebDNA receives a Replace command, it immediately searches for the specified records in the database and replaces those records' fields with the named field values.

SQL/ODBC Note: To replace records in an ODBC-compliant table, use the [SQL] context.

For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=Replace&db=SomeDatabase.db&eqNAMEdata=Grant&name=John&address=1492%20Somewhere%20Lane&zip=90000>

The database "SomeDatabase.db" opens and all records whose name field is "Grant" are found. The field names "name" is set to "John," "address" is set to "1492 Somewhere Lane," and "zip" is set to "90000."

Any fieldnames not existing in the database are ignored, and any fields you do not specify are left blank in the new record. Certain letters are illegal, such as <tab> or <carriage return>, so they are converted to <space> and <soft return> before being added to the database. Some computers use the two-character sequence <carriage return><line feed> to indicate a single end of line, which is converted automatically to a single <soft return> character before being added to the database.

You may specify a relative or full URL to the database file, as in “/WebDNA/Folder/SomeDatabase.db” or “../SomeDatabase.db”. The path to the database is always relative to the template URL, so if the database is in the same folder as the template, the path would be “SomeDatabase.db”, and if the database was one folder up, then the path would be “../SomeDatabase.db”.

Note: You may force the visitor to enter something into a field by using the RequiredFields parameter in the URL. Setting RequiredFields=name+address +city displays an error message if the visitor forgets to enter text into any of those three fields. RequiredFields works for all commands, not just this one.

Several equivalent ways to send the same command include:

HTML Source	Description
<pre></pre>	Hyperlink command
<pre><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="Replace"> <input type="hidden" name="template" value="xx.tpl"> <input type="hidden" name="db" value="SomeDatabase.db"> <input name="eqNAMEdata" value="John"> <input name="name"> <input name="address"> <input type="submit"> </form></pre>	Form-based command

Parameters for Replace include:

Parameter	Description
Append	(Optional) “T” if you want a new record to be added to the end of the database in the case where no records were found to be replaced.

[REPLACEFOUNDITEMS] CONTEXT

Syntax: [ReplaceFoundItems]field1=value1&field2=value2[/ReplaceFoundItems]

Result: Replaces each found record in a database with the new field values.

SQL/ODBC Note: To replace records in an ODBC-compliant table controlled by a SQL server, use the [SQL] context.

To replace field values of records in a database, put a [ReplaceFoundItems] context into a template inside a [Search] context. As each matching record is found, that record's fields inside the [ReplaceFoundItems] context are replaced with new values.

Note: This new context is much faster than the old technique of nesting a [Replace] context inside a [FoundItems] context. For example: if you currently use something like this to modify many records in a database...

```
[Search db=xx.db&neSKUdata=0]
  [FoundItems]
    [Replace
      db=xx.db&eqSKUdata=[sku]]value=[math][value]+1[/math][[/Replace]
  [/FoundItems]
[/Search]
```

then you can change it to the following in order to speed it up considerably:

```
[Search db=xx.db&neSKUdata=0]
  [ReplaceFoundItems]value=[math][value]+1[/math][[/Replace]
[/Search]
```

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[Search db=products.db&neSKUdata=0]
  [ReplaceFoundItems]price=[math][price]*1.1[/math][[/ReplaceFoundItems]
[/Search]
```

In the example above, the database "products.db" opens, all records whose sku field is not "0" found, and each of those found record's price fields incremented by 10%. As each found record is visited, that record's field values are available inside the context so you can use them to compute new values. This behavior is very different from the simpler [Replace] context, which replaces all found items with the same value.

Any fieldnames that do not exist in the database are ignored, and if you leave some existing fieldnames out of the replace context, they will remain unchanged in the database. Certain letters are illegal, such as <tab> or <carriage return>, so they are converted to <soft tab> and <soft return> before being added to the database. Some computers use the two-character sequence <carriage return><line feed> to indicate a single end of line, which

is automatically converted to a single <soft return> character before being added to the database. These 'soft' characters are automatically converted back to 'hard' versions (the originals) whenever you retrieve fields from a search of the database.

You may specify an absolute or relative path to the database file, as in "/WebDNA/GeneralStore/SomeDatabase.db" or "../SomeDatabase.db". You may also place "^" in front of the database path to indicate that the file can be found in a global root folder called "Globals" inside the WebCatalogEngine folder.

Contrast between [ReplaceFoundItems] and [Replace]

[Search db=products.db & neSKUdata=0] [ReplaceFoundItem]price=[math] [price]*1.1[/math] [/I eplaceFoundItems] [/Search]			[Replace db=products.db&neSKUdata=0] price=10 [/Replace]		
SKU	Price Before	Price After	SKU	Price Before	Price After
1	5	5.5	1	5	10
2	10	11	2	10	10
3	15	16.5	3	15	10
4	20	22	4	20	10
5	35	38.5	5	35	10

[SQL] CONTEXT

Syntax: [sql dsn=ODBC data source&statement=sql text]founditems[/sql]

Result: Performs a SQL statement on an ODBC data source.

Required Tag Parameters:

- **dsn=ODBC data source** – Name of the database you wish to search
- **statement=text** – Any legal SQL statement. Consult an SQL reference manual for more information.

- **max=number** – A number indicating how many records should be displayed at once before showing a list of “Show Items xx-yy” hyperlinks.

Optional Tag Parameters:

- **username=text** – Username required to access this ODBC database (if none specified, “sa” is used).
- **password=text** – Password required to access this ODBC database (if no is specified, a blank password is used).

Optional Context Parameters

- **[NumFound]** – A number indicating how many records matched the search request. Some ODBC drivers do not support this feature, so WebDNA compensates by visiting every record in the database in order to count them. For large datasets, this can be very slow, and you should consider writing a SQL statement that performs a count instead. Do not put [NumFound] inside an SQL context that inserts new records because the statement will be executed twice in order to perform the count. This will cause an invalid second record to be added to your database.
- **[FoundItems]...[/FoundItems]** – Normally a [FoundItems] loop is placed inside an [SQL] context that has performed a SELECT statement. This is done so that you can display all the matching records. You can insert any database field names inside the [FoundItems] loop to display them in the HTML.

To search through an ODBC-compliant database (or add new records, or delete or replace records,) place an [SQL] context into a WebDNA template. You may specify any DSN (Data Source Name) that has been properly configured through the ODBC setup control panel on the web server computer.

The [SQL] context is not limited to searching—you may perform any legal SQL statement, such as SELECT, INSERT, DROP, etc. The SQL language is too broad to describe here; it is assumed you have a working knowledge of SQL before using this context.

For example:

```
[SQL dsn=Pubs&statement=SELECT * from Authors]
Found [NumFound] items<br>
[FoundItems]
[au_lname], [au_fname], [title]<br>
```

[/FoundItems]
[SQL]

Whenever WebDNA encounters an [SQL] context, it uses ODBC to attempt to make a connection to the specified DSN. It then executes the SQL statement and retrieves the results, if any. For SQL SELECT statements, you almost always place a [FoundItems]...[/FoundItems] context inside the [SQL] context so you can display the information from the matching records.

Shopping

ADD COMMAND

Syntax: Add?db=DatabaseName&cart=[cart]&sku=xx&quantity=xx

Adds a product to the specified shopping cart.

To add products to a visitor's shopping cart, click a URL containing the Add command (alternately, you may embed an [AddLineItem] context into a template). Whenever WebDNA receives an Add command, it opens the shopping cart file (creating a new one if necessary) and adds the product (identified by its SKU) to the end of the LineItems in the shopping cart. The item's price, taxable, canEmail, and unitShipCost information is found by looking for the values of those fields in the product database. You can use a different price by creating a Formulas.db database. Also see Remove, Clear, ShowCart, and Purchase.

For example, normally you would link to a URL or form containing the following information:

<http://yourserver.com/xx.tpl?command=Add&db=SomeDatabase.db&sku=1234&cart=5678&quantity=5>

The database "SomeDatabase.db" opens, and sku 1234 is found. Shopping Cart file "5678" opens, and a new line item is added to the bottom of the list. The item's quantity is 5 (as specified in the command above), and the price is taken from the database's price field (or, if a formula for [price] is available in Formulas.db, the price is calculated using that formula). The page sent back to the browser will be xx.tpl, which typically contains a [LineItems] loop to display the current items in the cart.

Note: normally all database file paths are relative to the local template. If they begin with "/" they are relative to the web server's virtual host root. As of

version 3.0, you may optionally put a “^” in front of the file path to indicate the file can be found in a global root folder called “Globals” inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

Note: you may add a maximum of 100 lineitems to a shopping cart.

Here are the parameters to the Add command:

Parameter	Description
db	Product database that contains the SKU, price, and other information
Sku	Uniquely identifies which product should be added to the cart.
Cart	Shopping cart file that is to be affected
password	(Optional) In order to change the price (see below) you must provide the lineitem change password, which can be set in the preferences.
Optional Parameters	Description
price	(Optional) Sometimes you may need to change the price of a product while adding it to the cart. Normally you use a formula to vary pricing, but sometimes this alternate technique is preferred. Remember to put the lineitem change password (see above) into the parameters. There is a security risk when using this technique, because outsiders can change the price to anything they like.
textA	(Optional) This is extra information of any kind associated with this line item. Often used to store extra product information, such as "shoe size" or "color." Also used to pass catalog database fields such as [title] through to the order file so they may be viewed later without needing the original database to look for the value of [title].
textB	(Optional) Same as textA above.
textC	(Optional) Same as textA above.
textD	(Optional) Same as textA above.
textE	(Optional) Same as textA above.

Parameter	Description
quantity	(Optional) Tells how many of this SKU should be added to the cart. This quantity is used when calculating subtotals, unitShipCost, etc.
taxable	(Optional) "T" or "F". Overrides "taxable" field in the database - normally the information about the item's taxable status is taken from a field called "taxable."
canEmail	(Optional) "T" or "F". Overrides "canEmail" field in the database - normally the information about the item's canEmail (electronically deliverable) status is taken from a field called "canEmail."
unitShipCost	(Optional) A number indicating the item's price for shipping. Overrides "unitShipCost" field in the database - normally the information about the item's unitShipCost status is taken from a field called "unitShipCost". ShipTotal and GrandTotal use this number (multiplied by quantity) to determine the total shipping and grand total.
Header Field	You may set any shopping cart header field (such as Name, taxRate, Address1, etc.) at the same time you add a product to the cart.
RequiredFields	You may force the visitor to enter something into a field by using the RequiredFields parameter in the URL. Setting RequiredFields=field1+field2+field3 will display an error message if the visitor forgets to enter text into any of those 3 fields. RequiredFields works for all commands, not just this one.

[ADDLINEITEM] CONTEXT

Syntax: [AddLineItem Parameters]values[/AddLineItem]

Adds a product to the specified shopping cart.

To add products to a visitor's shopping cart, place an AddLineItem context into a template. (Note that you may also use the Add command from a URL

or a FORM.) Whenever WebDNA encounters an AddLineItem context, it opens the shopping cart file (creating a new one if necessary) and adds the product (identified by its SKU) to the end of the LineItems in the shopping cart. The item's price, taxable, canEmail, and unitShipCost information is found by looking for the values of those fields in the product database. A different price can be used by creating a Formulas.db database. For additional uses, see the following: Remove, Clear, ShowCart, [SetLineItem] and Purchase.

For example: The following text is usually placed into a .tpl file on your server and uses a web browser to link to it):

```
[AddLineItem cart=5678&sku=1234&db=catalog.txt]
  quantity=5&textA=Red[/AddLineItem]
```

Note: Line items can also be added to order files not inside the ShoppingCarts folder. By using file=/folder/folder/cartname instead of cart=cartname, you can affect any order file in any folder. This is most often used for "back door" administrative maintenance by designated personnel only.

Note: A maximum of 100 line items can be added to a shopping cart.

The database "catalog.txt" opens, and sku 1234 is found. Shopping cart file "5678" opens, and a new line item is added to the bottom of the list, with a quantity of 5 and textA set to "Red" (as specified in the context above). The price is taken from the database's price field (or, if a formula for [price] is available in Formulas.db, the price is calculated using that formula).

The parameters to the AddLineItem context are:

Parameter	Description
Db	Product database containing the SKU, price, and other information.
Sku	Uniquely identifies which product should be added to the cart.
Cart	Affected shopping cart file (from ShoppingCarts folder)
File	(alternative to cart) Alternate affected shopping cart file (from any folder). Unlike cart, this file can be in any folder. Specify the file URL-relative to

	the template.
--	---------------

Context Values	Description (values are inside the Context)
Password	(Optional) In order to change the price (see below) you must provide the line item change password, which can be set in the preferences.
Price	(Optional) Sometimes you may need to change the price of a product while adding it to the cart. Normally you use a formula to vary pricing, but sometimes this alternate technique is preferred. Remember to put the line item change password into the parameters. There is a security risk when using this technique, because outsiders can change the price to anything they like.
TextA	(Optional) Extra information of any kind that you want associated with this line item. Often used to store extra product information, such as "shoe size" or "color." Also used to pass catalog database fields such as [title] through to the order file so they may be viewed later without needing the original database to look for the value of [title].
TextB	Same as textA above.
TextC	Same as textA above.
TextD	Same as textA above.
TextE	Same as textA above.
Quantity	(Optional) Indicates how many of this SKU should be added to the cart. This quantity is used when calculating subtotals, unitShipCost, etc.
Taxable	(Optional) "T" or "F". Overrides "taxable" field in the database - normally the information about the item's taxable status is taken from a field called "taxable."
CanEmail	(Optional) "T" or "F". Overrides "canEmail" field in the database - normally the information about the

Context Values	Description (values are insic ɔ the Context)
	the database - normally the information about the item's canEmail (electronically deliverable) status is taken from a field called "canEmail."
UnitShipCost	(Optional) A number indicating the item's price for shipping. Overrides "unitShipCost" field in the database – normally the information about the item's unitShipCost status is taken from a field called "unitShipCost." ShipTotal and GrandTotal use this number (multiplied by quantity) to determine the total shipping and grand total.
Header Field	You may set any shopping cart header field (such as Name, taxRate, Address1, etc.) at the same time you add a product to the cart.

[CART] TAG

Syntax: [Cart]

Placing [Cart] in your template automatically creates a unique shopping cart identifier that can be used in eCommerce commands such as [Add](#), [Remove](#), [Purchase](#), etc. If no cart value is specified when you arrive at a page, then a new unique value is created. If you pass a cart value into the URL or form (propagating the same value from page to page), then that same cart value is used throughout the page.

[Cart] is also handy for times when you just need a guaranteed-unique value somewhere, such as a record identifier or even a product SKU. For this reason, [Cart] works in Typhoon as well as WebDNA.

CLEAR COMMAND

Syntax: Clear?cart=[cart]

Result: Removes all products from the specified shopping cart.

To remove all products from a visitor's shopping cart, click a URL containing the Clear command. Whenever WebDNA receives a Clear command, it

opens the shopping cart file and removes all LineItems from the shopping cart. The template displayed after the Clear command can contain any header fields from the order file (cart), and can contain a [LineItems] loop (but of course no line items ever appear because they are all gone).

For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=Clear&cart=5678>

Shopping Cart file "5678" opens, and all line items are removed from the list. The page sent back to the browser is xx.tpl, which typically contains a message confirming the shopping cart is empty.

Parameters for the Clear command include:

Parameter	Description
Cart	Shopping cart file to be cleared
Template	Template of HTML to be displayed after clearing

[CLEARLINEITEMS] TAG

Syntax: [ClearLineItems cart=cartID]

Placing [ClearLineItems] in your template will remove all line items from the specified shopping cart. Normally carts are found inside the ShoppingCarts folder, but you may specify a cart in any folder by using file=/folder/cartID instead of cart=cartID.

[LINEITEMS] CONTEXT

Syntax: [lineitems]LineItem Variables[/lineitems]

Result: Loops through all the line items in an order file.

Optional Context Tags:

- **[lineindex]** – A number from 1 to the number of line items, indicating this item's index in the list.
- **[sku]** – The SKU of this lineitem in the shopping cart.

- **[quantity]** – Quantity of this lineitem.
- **[price]** – Price of this lineitem.
- **[taxable]** – “T” if this item is taxable, “F” if not.
- **[canemail]** – “T” if this item is electronically deliverable, “F” if not.
- **[unitshipcost]** – Price to ship one unit of this item.
- **[texta]** – Extra text field to be used for any purpose. Often used to store extra product information, such as “shoe size” or “color”. Also used to pass catalog fields such as [Title] through to the order file.
- **[textb]** – Used in the same way as [texta].
- **[textc]** – Used in the same way as [texta].
- **[textd]** – Used in the same way as [texta].
- **[texte]** – Used in the same way as [texta].

To display a list of all the line items in a shopping cart or order file, insert a [LineItems] context into a template. This is typically in a template that displays the results of a showCart, add, remove, or purchase command. You may also put a [LineItems] context inside an [OrderFile] context embedded in any page.

For example:

```
[lineitems]
[sku], [price], [quantity]<br>
[/lineitems]
```

NEWCARTSEARCH COMMAND

Syntax: NewCartSearch?SearchParameters (Deprecated)

Searches, then creates a new, unique shopping cart token and searches a database after substituting [cart] tags. Deprecated is no longer needed because [cart] automatically generates a new cart number when none is specified.

To create a new shopping cart token, send WebDNA a NewCart command with the name of the template file you want to display.

Whenever WebDNA receives a NewCart command, it immediately creates a unique cart number, opens the specified file, looks for and interprets any [xxx] tags, and displays the results to the visiting web browser.



This command is obsolete and is provided for backward-compatibility only. Instead, use an embedded [Search] context.

WebDNA's "SmartCart" feature automatically creates a shopping cart token if it detects a [cart] tag in any page you display without explicitly sending the NewCart command. Consequently, simply linking to a page with [cart] tags will work correctly.

For example, normally you would link to a URL or form containing the following information:

<http://yourserver.com/xx.tpl?command=NewCartSearch&eqNamedata=Grant>

NEWCART COMMAND

Syntax: NewCart

Result: Creates a new, unique shopping cart token and displays an HTML text file after substituting [cart] tags. Deprecated is no longer necessary because [cart] automatically generates a new cart number when no cart is specified.

Note: This command is no longer supported as of Version 4.0.

To create a new shopping cart token, send WebDNA a NewCart command with the name of the template file you want to display. Whenever WebDNA receives a NewCart command, it immediately creates a unique cart number, opens the specified file, looks for and interprets any [xxx] tags, and displays the results to the visiting web browser.



This command is obsolete and is provided for backward-compatibility only. WebDNA's "SmartCart" feature automatically creates a shopping cart token if it detects a [cart] tag in any page you display without explicitly sending the NewCart command. Consequently, simply linking to a page with [cart] tags will work correctly. For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=NewCart>

[ORDERFILE] CONTEXT

Syntax: [orderfile cart=cartID]OrderFile Tags[/orderfile] or
[orderfile file=filepath][[/orderfile]

Result: Displays the contents of an order file or shopping cart.

Optional Context Tags: `[lineitems]/[lineitems]` (Loops through all the line items in the order file / shopping cart.)

Header Tags:

- **[payMethod]** – Payment method chosen by customer. May be one of: CC - Credit Card, AC - Account number (purchase order) ,FV - First Virtual.
- **[accountNum]** – Account number for payment. May be credit number, purchase order, or First Virtual account.
- **[expMonth]** – Month the credit card expires.
- **[expYear]** – Year the credit card expires.
- **[email]** – Email address the customer entered into the invoice page.
- **[name]** – Customer's name.
- **[company]** – Customer's company.
- **[address1]** – First address line entered by customer for shipment.
- **[address2]** – Second address line entered by customer for shipment.
- **[city]** – Customer's city.
- **[state]** – Customer's state.
- **[zip]** – Customer's zip code.
- **[country]** – Customer's country.
- **[phone]** – Customer's phone number.
- **[shipToEmail]** – Ship-To email address the customer entered into the invoice page.
- **[shipToName]** – Ship-To customer's name.
- **[shipToCompany]** – Ship-To customer's company.
- **[shipToAddress1]** – Ship-To first address line entered by customer for shipment.
- **[ShipToAddress2]** – Ship-To second address line entered by customer for shipment.
- **[shipToCity]** – Ship-To customer's city.
- **[shipToZip]** – Ship-To customer's state.

- **[shipToCountry]** – ShipTo customer’s country.
- **[shipToPhone]** – Ship-To customer’s phone number.
- **[taxRate]** – Percentage tax rate for this order, such as 7.75.
- **[shipVia]** – Method of delivery. May be one of: EMAIL - Electronic delivery via email attachments, WEB - Electronic delivery via automatically-built web pages, FedEx - Any other text can be entered here. Has no special meaning, but can be passed through to fulfillment emails. Maximum of 14 characters.
- **[header1]** – Any extra information you want to keep in the order file, up to 255 characters.
- **[header2] - [header40]** – Same as header1.

Calculated Values:

- **[subTotal]** – Total of the quantity * price for all the lineitems.
- **[taxableTotal]** – Total of the quantity * price for line items whose taxable field is set to true (“T”).
- **[taxTotal]** – TaxableTotal * taxRate defined in the header.
- **[shipTotal]** – Total of the unitShip cost * quantity for all line items plus the baseShipCost.
- **[grandTotal]** – Total cost for the order: includes tax and shipping.

To display the contents of an order file (either a shopping cart or a completed order), insert an [orderfile] context into the template. You may insert a [LineItems] context inside the [OrderFile] context to display each of the line items (products) in the order file. Since the [OrderFile] . . . [/OrderFile] context can specify a file by either the its path or its [Cart] number, it can be used in place of the ShowPage command.

For example:

```
[orderfile Orders/-13490876]
Name: [name]<br>
Address: [address1]<br>
[address2]<br>
Grand Total: [grandTotal]
[/orderfile]
```

[PURCHASE] TAG

Syntax: [Purchase **cart**=cartID]

Placing [Purchase **cart**=cartID] in your template moves the specified shopping cart file from the ShoppingCarts folder to the Orders folder, effectively the same as a Purchase command. If the cart file is not in a ShoppingCarts folder, you may use the alternate **file**=/folder/cartID instead of **cart**=cartID.

PURCHASE COMMAND

Syntax: Purchase?db=xx.db&cart=123&options=xx

Result: Submits a shopping cart for final order processing.

To purchase a shopping cart full of items, send WebDNA a Purchase command. WebDNA verifies the checksum of the credit card (if paying by credit card), sets any quantities, taxRates, or other order file header variables, and moves the order file from the Shopping Cart folder to the Orders folder. The template displayed is usually a “Thank You” template, which can contain [Email] contexts to send order verification and fulfillment requests to people in your company. The “Thank You” template can also contain any header variables from the cart, as well as a [LineItems] context so you can display final grand totals and other verification of the items purchased.

WebDNA performs credit card validation or fulfillment of the order if you are using the WebMerchant feature. WebDNA performs a simple check of the credit card number to see if it is a reasonable number, but it does not verify funds.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

<http://www.yourserver.com/GeneralStore/ThankYou.tpl?command=Purchase&db=xx.db&cart=123>

Note: As a rule, all database file paths are relative to the local template, or if they begin with “/,” they are relative to the web server’s virtual host root. You may optionally put “^” in front of the file path to indicate the file can be found in a global root folder called “Globals” inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

You may set any of the following variables during a Purchase command by placing named form variables into the purchase form. Some are optional, and any you do not explicitly set will remain unchanged from their previous values obtained from Add or ShowCart commands.

Parameter	Description
Template	(Required) The template displaying the "Thank You" page after the purchase command completes. If you are using suffix mapping, then the template is simply the URL of the "Thank You" page.
db	(Required) Product Database use for looking up product [xxx] fields inside the [LineItems] loop.
Cart	(Required) Name of the shopping cart file to be purchased.
PayMethod	(Required) Payment method chosen by customer. May be one of the following: <ul style="list-style-type: none"> . CC - Credit Card (WebDNA performs a checksum to make sure the number is reasonable) . AC - Account number (purchase order) (WebDNA performs no validation, but WebMerchant Account Authorizer later does custom authorization) . BK - Book this order (verify funds, but do not deposit money). Handled by WebMerchant . SH - Ship this order (only from a previous BK order. Deposits money that was verified earlier). Handled by WebMerchant
AccountNum	(Required) Account number for payment. May be credit card number, purchase order, or First Virtual account.
ExpMonth	(Required for credit cards) Month the credit card expires (between 1 and 12).

Parameter	Description
ExpYear	(Required for credit cards) Year the credit card expires.

Optional Parameters	Description
Card	<p>(Optional) If you only accept certain types of credit card, then set this to a list of card names you accept. For example: VISA+MC will accept VISA and MasterCard, but no others. You may choose from VISA, MC, AMEX, DISC, JCB, CB, DINER, ALL, IGNORE.</p> <ul style="list-style-type: none"> . ALL - means accept all cards listed above (if checksum is correct) . IGNORE - means accept any number, regardless of checksum.
Email	Bill-To: Email address the customer entered into the invoice page. Limited to 99 characters.
Name	Bill-To: Customer's name. Limited to 99 characters.
Company	Bill-To: Customer's company. Limited to 99 characters.
Address1	Bill-To: First address line entered by customer for shipment. Limited to 99 characters.
Address2	Bill-To: Second address line entered by customer for shipment. Limited to 99 characters.
City	Bill-To: Customer's city. Limited to 49 characters.
State	Bill-To: Customer's state. Limited to 49 characters.
Zip	Bill-To: Customer's zip code. Limited to 19 characters.

Optional Parameters	Description
Country	Bill-To: Customer's country. Limited to 49 characters.
Phone	Bill-To: Customer's phone number. Limited to 24 characters.
ShipToEmail	Ship-To: Email address the customer entered into the invoice page. Limited to 99 characters.
ShipToName	Ship-To: Customer's name. Limited to 99 characters.
ShipToCompany	Ship-To: Customer's company. Limited to 99 characters.
ShipToAddress1	Ship-To: First address line entered by customer for shipment. Limited to 99 characters.
ShipToAddress2	Ship-To: Second address line entered by customer for shipment. Limited to 99 characters.
ShipToCity	Ship-To: Customer's city. Limited to 49 characters.
ShipToZip	Ship-To: Customer's zip code. Limited to 19 characters.
ShipToState	Ship-To: Customer's state. Limited to 19 characters.
ShipToCountry	Ship-To: Customer's country. Limited to 49 characters.
ShipToPhone	Ship-To: Customer's phone number. Limited to 24 characters.
TaxRate	Percentage tax rate for this order, such as 7.75.
ShipVia Method of delivery.	<p>May be one of:</p> <p>EMAIL - Electronic delivery via email attachments</p> <p>WEB - Electronic delivery via automatically-built web pages</p> <p>FedEx - Any other text can be entered here. Has no special meaning, but can be passed through to</p>

Optional Parameters	Description
	fulfillment emails. Maximum of 63 characters.
header1	Any extra information you want to keep in the order file, up to 255 characters.
header2 - header40	Same as header1.
NonTaxableTotal	Normally this value is computed automatically, but if you explicitly set its value, you can 'override' the pre-computed value with any new number you like. WebMerchant displays this value if it is explicitly set, otherwise it computes it from the numbers provided in the order file.
TaxableTotal	Similar to nonTaxableTotal.
TaxTotal	Similar to nonTaxableTotal.
ShippingTotal	Similar to nonTaxableTotal.
SubTotal	Calculated: based on taxableTotal + nonTaxableTotal. This cannot be overridden directly; if you want to override the subTotal of an order, you must override both taxableTotal and nonTaxableTotal to force the calculation of subTotal to become a new value.
CartIPAddress	This is not pre-set automatically, but you can set it to [ipAddress] if you want to remember the IP address of the visitor who last used the cart. Limited to 15 characters.
CartUsername	You can store the user name of the person who last used this cart here. It is not set automatically. Limited to 63 characters.
CartPassword	You can store the password of the person who last used this cart here. It is not set automatically. Limited to 63 characters.
Precision	International support: number of digits after the decimal that are considered important for tax,

Optional Parameters	Description
	subtotal, grandtotal calculations. In the U.S., 2 digits are the normal precision. For Japan, 0 causes all calculations to round off at integer values. You can change this on a cart-by-cart basis with [SetHeader]
TaxableShipping	If set to T, causes shipping to be taxed. The tax and grand total are increased by [taxRate]*[shippingTotal]. Default value is F.
AuthNumber	WebMerchant Only: Text authorization ticket# returned from bank network. WebMerchant fills in this field after calling the bank. Limited to 63 characters.
ResponseText	WebMerchant Only: Response text returned from bank network. WebMerchant fills in this field after calling the bank. Limited to 149 characters.
Status	WebMerchant Only: status of order, such as Approved/Hold/Call/Pending. WebMerchant fills in this field after calling the bank. Limited to 63 characters.
BatchNumber	WebMerchant Only: Batch# that PCAuthorize/MacAuthorize has stored this order in. WebMerchant fills in this field after calling the bank. Limited to 63 characters.
ReferenceNumber	WebMerchant Only: Reference# that PCAuthorize/MacAuthorize created for this order. WebMerchant fills in this field after calling the bank. Limited to 63 characters.
SequenceNumber	WebMerchant Only: Sequence# that PCAuthorize/MacAuthorize created for this order. WebMerchant fills in this field after calling the bank. Limited to 63 characters.
ItemNumber	WebMerchant Only: Item# (within the batch) that PCAuthorize/MacAuthorize created for this order. WebMerchant fills in this field after calling the bank. Limited to 63 characters.

Optional Parameters	Description
RequiredFields	You may force the visitor to enter something into a field by using the RequiredFields parameter in the URL. Setting RequiredFields=field1+field2+field3 displays an error message if the visitor forgets to enter text into any of those 3 fields. RequiredFields works for all commands, not just this one.

Line Item Variables	Description
quantity[lineIndex]	You may optionally set the quantity (to be purchased) of any line item. For example: quantity1=12 sets the first line item's quantity to 12. Similarly, quantity5=3 sets the fifth line item's quantity to 3 (these quantities can also be set with a Add or ShowCart command)
textA[lineIndex]	Any extra text you wish to associate with this line item. Sometimes used to store size, color, or pass catalog fields through to the shopping cart (can also be set with Add or ShowCart command). Limited to 255 characters.
textB[lineIndex]	Similar to textA.
textC[lineIndex]	Similar to textA.
textD[lineIndex]	Similar to textA.
textE[lineIndex]	Similar to textA.
taxable[lineIndex]	You are not allowed to set this value. It comes from the product database field "taxable," or from the taxable formula in Formulas.db.
canEmail[lineIndex]	Similar to taxable unitShipCost[lineIndex]. To prevent "hacking" this value cannot be set remotely—its value either comes from the product database field "unitShipCost", or is computed from a formula stored in Formulas.db.

unitShipCost[lineIndex]	To prevent "hacking" this value cannot be set remotely -- its value either comes from the product database field "unitShipCost", or is computed from a formula stored in Formulas.db.
price[lineIndex]	Similar to unitShipCost.

[REMOVELINEITEM] TAG

Syntax: [RemoveLineItem **cart=cartID&index=3**]

Placing [RemoveLineItem] in your template immediately deletes the specified line item from the specified shopping cart file. Alternately, you may use the Remove command. See also [AddLineItem] and Add. If the cart file is not in a ShoppingCarts folder, you may use the alternate form **file=/folder/cartID** instead of **cart=cartID**.

REMOVE COMMAND

Syntax: Remove?db=DatabaseName&cart=[cart]&sku=xx

Removes a product from the specified shopping cart.

To remove products from a visitor's shopping cart, click a URL containing the Remove command. Whenever WebDNA receives a Remove command, it opens the shopping cart file and removes the product (identified by its SKU) from the LineItems in the shopping cart. Also see Add, Clear, ShowCart, and Purchase.

For example, normally you would link to a URL or form containing the following information:

<http://yourserver.com/xx.tpl?command=Remove&db=SomeDatabase.db&sku=1234&cart=5678>

Shopping Cart file "5678" opens, and the first line item found with the correct SKU is deleted. The page sent back to the browser will be xx.tpl, which typically contains a [LineItems] loop to display the current items in the cart (after the removal).

Here are the parameters to the Remove command:

Parameter	Description
db	Product database containing the SKU, price, and other information
sku	Uniquely identifies the product to remove from the cart
cart	Affected shopping cart file
template	Template of HTML displayed after the sku is removed from the cart. Typically this is the same shopping cart template used for adding items to the cart—so the visitor can see the item has been removed.

[SETHEDER] CONTEXT

Syntax: [setheader cart=cartID&index=x]line item values[/setheader]

Result: Changes a line item in a shopping cart.

Required Tag Parameters:

- **db=databasepath** – URL-style path to the product database.
- **index =number** – Uniquely identifies which line item should be modified.
- **cart=cartID** – Shopping cart file (from shopping carts folder) that is to be affected.

or

file=filepath – URL-style path to the shopping cart or order file that is to be affected. Unlike cart, this file can be in any folder.

Optional Context Parameters:

- **Email=text** – Email address of the person ordering.
- **PayMethod="CC" or "AC" or "FV"** – CC = Credit Card, AC = Account, FV = First Virtual (Pending Order).
- **AccountNum=text** – Credit card number, account number, First Virtual PIN etc.
- **ExpMonth=number** – 1=January, 2=February and so on.

- **ExpYear=number** – Either two digit (normally) or 4 digit year. Depends upon your credit card clearing software.
- **TaxRate=number** – Decimal value like .07 = 7% tax.
- **Name=text** – Name of person ordering.
- **Company=text** – Company of person ordering.
- **Address1=text** – First address line of person ordering.
- **Address2=text** – Second address line of person ordering.
- **City=text** – City of person ordering.
- **State=text** – State of person ordering.
- **Zip=text** – Zip code of person ordering.
- **Phone=text** – Phone number of person ordering.
- **ShipVia=text** – Shipping method like “Ground”.
- **ShipCost=number** – Costing of shipping chosen in ShipVia.
- **Country=text** – Country of person ordering.
- **ShipToEmail=text** – Email address of the person receiving the product.
- **ShipToName=text** – Name of the person receiving the product.
- **ShipToCompany=text** – Company name of the person (company) receiving the product.
- **ShipToAddress1=text** – First address of the person receiving the product.
- **ShipToAddress2=text** – Second address of the person receiving the product.
- **ShipToCity=text** – City of the person receiving the product.
- **ShipToState=text** – State of the person receiving the product.
- **ShipToZip=text** – Zip code of the person receiving the product.
- **ShipToCountry=text** – Country of the person receiving the product.
- **ShipToPhone=text** – Phone number of the person receiving the product.
- **Header1 - Header40=text** – Miscellaneous header fields where you can store information.

Although many of the header fields have specific names (e.g., “ShipToCompany”) you may use them for whatever data you wish. However, if you are using WebMerchant, some of the header fields must contain proper information. Those fields are in bold. You may also set header fields when conducting a SetLineItem or Purchase. Use SetLineItem to set the line item values in a shopping cart.

[SETLINEITEM] CONTEXT

Syntax: [setlineitem cart=cartID&index=x]line item values[/setlineitem]

Result: Changes a line item in a shopping cart.

Required Tag Parameters:

- **db=databasepath** – URL-style path to the product database.
- **index =number** – Uniquely identifies which line item should be modified.
- **cart=cartID** – Shopping cart file (from shopping carts folder) that is to be affected.

or

file=filepath – URL-style path to the shopping cart or order file that is to be affected. Unlike cart, this file can be in any folder.

Optional Context Parameters:

- **textA=value** – Extra information of any kind that you want associated with this line item. Often used to store extra product information, such as “shoe size” or “color”. Also used to pass catalog database fields such as [Title] through to the order file.
- **textB=value** – Same as textA above.
- **textC=value** – Same as textA above.
- **textD=value** – Same as textA above.
- **textE=value** – Same as textA above.
- **quantity=number** – Tells how many of this SKU should be added for this line item. This quantity is used when calculating totals, unitShipCost, etc.

- **taxable=Boolean** – “T” or “F”. Overrides taxable field in the database - normally the information about the item’s taxable status is taken from a field called taxable.
- **canEmail=Boolean** – “T” or “F”. Overrides canEmail field in the database - normally the information about the item’s canEmail (electronically deliverable) status is taken from a field called canEmail.
- **unitShipCost=number** – A number indicating the item’s price for shipping. Overrides unitShipCost field in the database - normally the information about the item’s unitShipCost status is taken from a field called unitShipCost. ShipTotal and GrandTotal use this number (multiplied by quantity) to determine the total shipping and grand total.

SHOWCART COMMAND

Syntax: ShowCart?db=xx.db&template=xx.tpl&cart=[cart]&options=xx

Displays/Modifies the contents of a shopping cart order file.

To display a shopping cart full of items (or update some header or line item information in the cart), send WebDNA a ShowCart command. WebDNA looks for any new values of header fields in the form and sets the corresponding fields in the cart. It also looks for any numbered line item information, such as "quantity3=39" and modifies the corresponding line item values in the cart. See Purchase and Add.

For example, normally you would put the following text into a .tpl file on your server and use a web browser to link to it:

<http://www.yourserver.com/WebDNA/ShoppingCart.tpl?command=ShowCart&cart=123&db=catalog.txt>

You may set any of the same variables as with a Purchase command by putting named form variables into the form. Some are optional, and any you do not explicitly set remain unchanged from their previous values obtained from Add or ShowCart commands.

Any formulas you defined will be applied and calculated before showing the contents of the cart.

Note: You may force the visitor to enter something into a field by using the RequiredFields parameter in the URL. Setting RequiredFields=field1+field2+field3 displays an error message if the visitor forgets to enter text into any of those 3 fields. RequiredFields works for all commands, not just ShowCart.

In addition to the values you may set when displaying the cart, you may also display the following pre-calculated values:

Value	Description
GrandTotal	Calculated final purchase cost, based on price of all lineitems, quantities, base shipping cost, unit shipping cost, and taxRate.
SubTotal	Total cost of all items, before shipping and tax.

Value	Description
	and tax
TaxableTotal	Total cost of all items that are marked as taxable
TaxTotal	Tax on taxable items
ShippingTotal	Total cost of shipping based on unitShipCost of all items and base shipping charge.
NumLineItems	Number of line items in the order

[VALIDCARD] TAG

Syntax: [ValidCard **accountNum**=cardNumber&**card**=VISA+MC]

Placing [ValidCard] into a template displays “T” or “F” (True or False), depending on the value of the number in accountNum. If the accountNum is a reasonable credit card number for the specified bank network, then “T” indicates it is good. Conversely, “F” indicates it is bad. This **does not** call the bank to verify funds on the card—it merely does a simple numeric checksum to verify the number is consistent with a credit card number.

Parameters	Description
accountNum	(Required) The credit card number. Any extra spaces or non-numeric characters will be ignored.
card	(Optional) Set card to the names of the credit cards that are allowed, separated by space or +. Possible values are ALL, IGNORE, VISA, MC, AMEX, DISC, JCB, DINER . If not specified, then ALL cards are allowed. IGNORE tells it to always validate the card regardless of its checksum.

Showing and Hiding

[HIDEIF] CONTEXT

Syntax: [hideif comparison]Hide This HTML[/hideif]

Result: Hides HTML conditionally only if the comparison is true.

Required Tag Parameter: comparison (a comparison checks two pieces of text separated by a comparison operator)

The comparison is separated from the hideif in the beginning tag with a single space character. All other characters, including space characters, are considered part of the comparison. The characters representing the comparison operators are not valid in the text being compared.

Review the following table:

Comparison Type	Character	Example
equal	'='	[hideif [username]=SAGEHEN]
not equal	'!'	[hideif [random]!45]
contains	'^'	[hideif [browsername]^Mozilla]
begins with	'~'	[hideif [ipaddress]~245.078.013]
less than	'<'	[hideif [random]<50]
greater than	'>'	[hideif [lastrandom]>25]

If both side of the comparison are numbers, then the comparison for greater than, less than and equal is performed numerically. If either side is not a number, then the comparison is performed alphabetically.

To hide selected HTML (or [xxx] tags) only if certain conditions are met, insert the text inside a [hideif] container. The comparison, which may contain any [xxx] tags, is first evaluated to see if it is true, and if true then the contained text is hidden. If not true, then any text or [xxx] tags inside the container is displayed. See [ShowIf].

[hideif] functions appropriately when it hides its container: any contexts inside the [hideif] container (e.g. [append] or [replace] or [delete]) are not executed if [hideif] evaluates as “true.” A context must be completely enclosed within a single [hideif] context in order for it to be properly hidden. For example, you can’t have two [hideif] contexts that hide the beginning and ending tags for the [appendfile] context.

For example:

```
[hideif [username]=Grant]
[authenticate user Grant]
[/hideif]
```

[HTML1] CONTEXT

Syntax: [HTML1]Text for HTML 1 Browsers[/HTML1]

Result: Displays enclosed text only if the browser supports HTML 1 as defined in your Browser Info.txt file.

This context is used to display HTML and text only to older browsers that are listed in the Browser Info.txt file as belonging to this category.

For example:

```
[HTML1]
Any text that you want to be seen only
by older browsers that do not support tables or frames
[/HTML1]
```

In the example above, the displayed text is only the text you want to be seen by older browsers that do not support tables or frames (i.e., Netscape 1.0 or other non-table-aware browsers such as Lynx). You control which browsers see the HTML1 context by editing the Browser Info.txt file.

Both this and the following two contexts define how you can display different results to different browsers using a single template. Although we have named them after the three current levels of HTML, they offer more flexibility than simply displaying results fitting the definitions of HTML. By changing the three fields defined in the Browser Info.txt file, you can set these three contexts to display results based on any conditions you choose.

[HTML2] CONTEXT

See HTML1 Context.

[HTML3] CONTEXT

See HTML1 Context.

[IF][THEN][ELSE] CONTEXT

Syntax: [If Expressions][Then]do this[/Then][Else]otherwise this[/Else][/If]

Result: Displays HTML or executes WebDNA conditionally only if the expression is true.

To display HTML (or execute WebDNA [xxx] tags) only if certain conditions are met, place the text inside an [If] context. The comparison, which may contain any [xxx] tags, is first evaluated to see if it is true, and if true then the contained [Then] context is executed (or simply displayed, if it's just HTML). If not true, then the contained [Else] context is executed (or simply displayed, if it's just HTML). See [Then] and [Else].

Example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[If (("username"="Grant") | ([grandTotal]<100)) &
{{date}}<{2/15/2000})]
  [Then]either username was Grant or grandTotal was < $100 and it's not
  Feb 15, 2000 yet[/Then]
  [Else]The complex expression wasn't true[/Else]
[/If]
```

Comparisons are always case-insensitive so "grant" equals "GRANT". The expression is evaluated as a mathematical Boolean equation, where each sub-expression evaluates to either 0 or 1 (meaning true or false). If the entire evaluated expression is true, then the WebDNA inside the [Then] context is executed; otherwise, the [Else] context is executed. The [Math] context has been extended to allow for quoted text and Boolean operators, and is actually what is used by [If] to perform the work of evaluating the expression. A side-effect of this allows you to use these operators inside a [math] equation: [math]1<3[/math] evaluates to "1", because the equation is true. Conversely, [math]3<1[/math] evaluates to "0" because the equation is false. Similarly, [math]1&1[/math] evaluates to "1", and [math]1&0[/math] evaluates to "0".

Comparison		Example
Equal	=	[If "[username]" = "SAGEHEN"] variable [username] is equal to SAGEHEN
not equal	!	[If [random] ! 45] random number is not 45
contains	^	[If "[browsername]" ^ "Mozilla"] variable [browsername] contains the text Mozilla
begins with	~	[If "[ipaddress]" ~ "245.078.013"] variable [ipaddress] begins with 245.078.013 Notice the IP address has been typed with 3 digits in each portion of the address. This is very important for making these comparisons work as expected.
less than	<	[If [random] < 50] random number is less than 50
greater than	>	[If [lastrandom] > 25] last random number is greater than 25
divisible by	\	[If [index] \ 3] variable [index] is divisible by 3
or		[If (5>4) (1<3)] Boolean comparison: if either side of the operator is true, then the comparison is true
and	&	[If (5>4) & (1<3)] Boolean comparison: if both sides of the operator are true, then the comparison is true

Delimiter		Example
Quoted Text	".."	[If "Hello" ^ "hell"] All text must be surrounded by quotes

Delimiter		Example
Numbers		[If 12.5 < 13.2] Numbers do not need to be delimited; they function the same as in a [Math] context
Dates	{ }	[If {[date]} > {9/7/1963}] Dates must be enclosed in curly braces to distinguish them from regular numbers
Times	{ }	[If {[time]} > {12:31:00PM}] Times must be enclosed in curly braces to distinguish them from regular numbers
Parentheses	(..)	[If (3>1) & ("a"<"b")] You may collect groups of items in parentheses in order to force the order of evaluation

[SHOWIF] CONTEXT

Syntax: [showif comparison]Show This HTML[/showif]

Result: Displays HTML conditionally only if the comparison is true.

For example:

```
[showif [username]=Grant]You're allowed in![/showif]
```

See [Hidelf] for an explanation of comparison.

[SWITCH][CASE] CONTEXT

Syntax: [Switch Value]Series of [Case]...[/Case]contexts[/Switch]

Result: Executes the WebDNA inside the only [Case] context, which matches the given value.

To display HTML (or execute some WebDNA) from a list of known text options, put the text value inside a [Switch] context. For each possible option, put a [Case] context inside the [Switch]. You may optionally specify a default case by inserting a [Default] context. The [Default] context must be the very last context inside the [Switch].

For example, normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[text]x=5[/text]
[Switch value=[x]]
  [Case value=1]
    The value of x was 1
  [/Case]
  [Case value=2]
    The value of x was 2
  [/Case]
  [Default]
    The value of x was neither 1 nor 2; it was [x]
  [/Default]
[/Switch]
[text]title=Mrs[/text]
[Switch value=[title]]
  [Case value=Mr]
    You're a male
  [/Case]
  [Case value=Mrs]
    You're a female
  [/Case]
[/Switch]
```

In the first example above, the text “The value of x was neither 1 nor 2; it was 5” will display, because the two cases for “1” and “2” did not match the actual value of x, which was “5.” In the second example above, the text “You’re a female” will display. Any WebDNA inside the other [Case] contexts will not execute, and any text inside those contexts will not display.

Note: The values are compared as case-insensitive text only. This means the number “1.0” is not the same as the number “1” when determining which of the [Case] contexts to execute.

SHOWPAGE COMMAND

Syntax: ShowPage

Result: Displays an HTML text file after substituting [xxx] tags for their real values.

Note: This command will be supported through version 4.0. It will be eliminated in version 5.0.

To display an HTML file, send WebDNA a ShowPage command with the name of the template file you want to display. Whenever WebDNA receives a ShowPage command, it immediately opens the specified file, looks for and interprets any [xxx] tags, and displays the results to the visiting web browser.

If you have defined an ACTION telling your web server to send all files of a certain extension (.tpl, .HTML) to WebDNA, then simply linking to a URL with that file extension automatically informs WebDNA to perform a ShowPage on that template.

For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=ShowPage>

Other ways to send the same command include:

HTML Source	Description
<code></code>	Hyperlink to WebDNA (notice that when suffix-mapping for .tpl files is set to WebDNA's plug-in, WebDNA assumes all .tpl files should be displayed with the ShowPage command). You can set suffix mapping for .html, and then all files will be sent through WebDNA's interpreter.
<code><form method="POST" action="xx.tpl"> </form></code>	Form-based command (notice the template is part of the action, and ShowPage is automatic)

Security Note 1: Files inside your cgi-bin (or Scripts) directory cannot be displayed from a remote browser unless you explicitly create an extension mapping telling WebDNA to process them. This prevents outsiders from downloading order files or shopping cart files that contain sensitive information.

Security Note 2: Files whose type is 'WWWO' (where the O is the Macintosh character option-Z) will not be displayed. These files are considered to contain sensitive information that should never be shown to an outside user. WebDNA automatically creates files with this type in order to prevent hackers from downloading your sensitive data.

Dates and Times

[DATE] TAG

Syntax: [Date format=MM/DD/YYYY]

Placing [Date] in your template displays the current date as defined by the clock on your web server. Though the format of the date is normally MM/DD/YYYY, you can change the default format by changing the DateFormat preference. To override the default date format preference on a case-by-case basis you can specify a format inside the tag, as in [DATE %m/%d/%Y]. See [Time].

Note: The [Date] tag is sometimes confused with the [OrderFile]'s [Date] tag, which is not the same thing. The [Date] tag inside the context of an [OrderFile] represents the date the order was created, and does not have the ability to be formatted in special ways. If you want to display today's date inside of an [OrderFile], then you must first assign a text variable **outside** the OrderFile context, and use that text variable from then on.

The valid date formats are:

Format	Description
%a	Abbreviated weekday name "Wed" *
%A	Full weekday name "Wednesday" *
%b	Abbreviated month name "Feb" *
%B	Full month name "February" *
%c	Date and time in the form Wed Sep 19 18:24:21 1997 *
%d	Day of month 01-31
%H	Hour 00-23
%I	Hour 01-12

Format	Description
%j	Day of year 001-365
%m	Month 01-12
%M	Minute 00-59
%p	AM or PM
%S	Seconds 00-59
%U	Week # of year (where Sunday is first day of week) *
%w	Weekday 0 (Sunday) - 6 (Saturday)
%W	Week # of year (where Monday is first day of week) *
%x	Date as Sep 11 1997 *
%X	Time as 14:01:12
%y	Year without century 00-99
%Y	Year with century 1900-2199. Note: on Linux systems, years earlier than 999 will not display 4 digits for the year.
%Z	Time zone of server
%%	%
	* These items depend on the underlying operating system's Y2K-compliance. Windows NT, for instance, will not display correct day-of-week-name results for years past 2039. The workaround is to use %w (weekday as a number) and perform your own lookup in a small database for the textual name of the weekday.

[FORMAT] CONTEXT

Syntax: [Format formatspec]Text or Number[/format]

Result: Formats text or numbers in various widths, lengths, or currency formats.

To display numbers and text with various lengths, decimal points, or currency formats, surround the number or text with a [Format] context.

For example:

```
-[Format 10.2f]99.5[/Format]-
-[Format 14s]Hello, my name is John.[/Format]-
-[Format thousands .2f]394363210[/Format]-
```

```
-[Format thousands ,2f]394363210[/Format]-  
-[Format thousands .0f]394363210[/Format]-
```

Numbers are displayed right justified with enough preceding spaces and digits after the decimal point to fill the exact width of the format specifier. However, most browsers ignore preceding spaces unless you use a `<pre>` tag or similar method to display them. Text is left justified with enough spaces after it to exactly fill the width specifier. In the preceding examples, the hyphen character ‘-’ is used to show the resulting widths of the text displayed.

The results are:

```
- 99.50-  
-Hello, my name-  
-394,363,210.00-  
-394.363.210,00-  
-394,363,210-
```

The “f” specifier represents the floating-point number. Use this specifier to format numbers of any type. Numbers can be forced to a certain overall width and/or a certain number of characters after the decimal point. In addition, you can output numbers with commas (US style) or periods (European style) as separators using the thousands modifier before the format specifier.

Given a number 345.67, the following format specifiers will display as shown:

```
8.3f = - 345.670-  
8.2f = - 345.67-  
8.1f = - 345.7- (notice rounding from .67 to .7)  
.1f = -345.7-
```

The “s” specifier for strings is very useful for truncating long strings so they may be used in a synopsis (or lengthening short strings to fit some preformatted text). The “s” specifier must have a number in front of it specifying the width the text should be returned at (maximum length is 255). If your database contained technical support information in sentences or paragraphs, you might insert `[format 20s][solutiontext][format]` so that the first listing displays a short text snippet that could then be linked to the full detail record of the solution.

Dates and Times

For example:

```
[format days_to_date %d/%m/%Y]12/8/98[/format]  
[format seconds_to_time %H:%M]8:53:12[/format]  
[format days_to_date %d][math]{12/8/65}+10[/math][format]  
[format seconds_to_time %M][math]{10:53}-{10:45}[/format]
```

[/format]

You can also use the [format] context to format dates and time, or, perhaps more important, retrieve specific components of a date or time. Since the [math] context returns the results of any date operations as the number of days since 00/00/0000, and the results of any time operations as the number of seconds since midnight, the two format specifiers for [format] are `days_to_date` and `seconds_to_date`. After the specifier, use the ANSI-like formatting options described in the [time] and [date] tag sections to format the output.

The results of the examples above are:

8/12/1998 – Reformats from US-style dates to European-style dates.

8:53 – Truncates the time so it only displays hours and minutes.

8 – Returns the day value that's the result of the addition.

8 – Returns the number of minutes in the difference.

[MATH] CONTEXT

Syntax: [math]Equation[/math]

Result: Calculates the enclosed numerical, date, or time equation and displays the results.

Optional Tag Parameters:

- **date** – Displays the results of the calculations using the default date format in your preferences.
- **time** – Displays the results of the calculations using the default time format in your preferences.
- **show** – Possible values are “T” (true) or “F” (false) to display or suppress the results from appearing on the page.

Optional Context Parameters:

- **{date}** – Dates need to be enclosed in curly braces so they are not mistaken for multiple division expressions.
- **{time}** – Times, like dates, need to be enclosed in curly braces so they are not mistaken for numbers. Example: {11:23:56}.

- **+, -, *, /, %, ^** – All the standard math operators are supported.
- **(expression)** – Any math expression, including those involving dates and time can be enclosed in parentheses to force evaluation in a certain order.
- **variable=expression** – The [Math] context may be used to set and reset variables. Variables can have any text name. Their value is displayed by placing the variable name in square brackets.

To calculate a mathematical equation, put it inside a Math context. You can insert any [xxx] variables inside the context. Dates and times can also be calculated. To distinguish dates and times from plain numbers, put them inside curly-braces: {12/01/1997}. You cannot mix both dates and times in one equation.

You may also create up to 150 math variables by name. These named variables can be used in any other [Math] context in the template: [Math]fred=12/7.5[/Math] ...other text here...[Math]fred/15.2[/Math]. Note that the name of a math variable is limited to 15 characters in length and must begin with a letter followed by up to 14 letters or numbers. Only the letters a-z and A-Z are allowed.

For example:

```
[math](4.5+6.2)/17*95-12[/math]
[math]{4/7/1997}+10[/math]
[math]{4/7/1997}+{02/00/0000}[/math]
[math date]{4/7/1997}+10[/math]
[math date]{4/7/1997}+{02/00/0000}[/math]
[math date]{[date]}-{00/07/0000}[/math]
[Math]{12:51:02}[/Math]
[Math time]{12:51:02}+{01:00:05}[/Math]
[Math]x=5/3[/Math]
[Math]x=5%3[/Math] (% = Modulo Operator)
[Math show=F]xyz=5/3[/Math]
[Math]xyz[/Math]
[xyz] (3.0)
```

The resulting text displayed is:

```
47.7941176470588
729496 (4/7/1997 + 10 days expressed as number of days since
00/00/0000 (AD))
729547 (4/7/1997 + 2 months expressed as days since
00/00/0000)
```

```

04/17/1997 (4/07/1997 + 10 days expressed as date)
06/07/1997 (4/17/1997 + 2 months expressed as date)
03/30/1997 (One week ago today)
46262 (number of seconds between midnight and 12:51:02
expressed as seconds)
13:51:07 (12:51:02 pm plus 1 hour and 5 seconds expressed as
time)
1.66666666666667
2
(no output) show=F means not to display the results of the
equation
1.66666666666667 (simply naming a math variable inside a
math context displays its value)
1.66666666666667 (simply naming a math variable like [x]
displays its value)

```

Any [xxx] variables are first evaluated and replaced with their real values, then the resulting equation is calculated. The final numerical result is displayed. Standard algebraic order of operations are followed when evaluating expressions. Use parentheses to clarify or force a specific order of operations.

For example:

```

[lineitems]
[sku], [title], [description], [price], [math]([price]+[unitShipCost])*[quantity]/[math]
[/lineitems]

```

Dates

Dates may be included in mathematical expressions by enclosing the date in braces ('{' and '}'). You may easily add or subtract days, months, or years from dates by expressing them as a complete date. Use 0 (zero) for values to be ignored. For example, in order to add 2 months to today's date you would write an expression like: $\{[date]\}+{2/0/0000}$.

Keep the following in mind when using dates:

- It is a good idea to group math expressions involving dates together using parentheses.
- The year must be expressed as 4 digits so that 2-digit years can be converted to their proper value (i.e. 96 is really 1996, and 00 is 2000).
- When using dates mixed with integers, the final result is a value representing a number of days (i.e. $\{12/8/97\}+10$ adds 10 days to the date). In fact, the result of a math expression with dates is always the number of days. To display the output of the math expression as a

date, add the Date modifier to the [Math] context: [Math Date]...[/Math].

- **Non-American Dates (as of 3.0.3):** Some countries specify dates with decimal points, as in {10.1.1998}, but WebDNA will interpret this as a time instead. You can force it to interpret text as a Date by inserting a "D" in front of the text, as in [math]{D10.01.1998}/math], so 10.01.1998 is interpreted as a date instead of a time.

Times

Like dates, times are used in mathematical expressions by enclosing them in curly braces. You may easily add or subtract hours, minutes, or seconds from times by expressing them as a complete time. Use 0 for values that you want ignored. That is, in order to add 2 minutes to the current time you would write an expression like the following [math time][time]+{00:02:00}/math]. It is a good idea to group math expressions involving time together by using parentheses.

The result of an expression involving times is the number of seconds since midnight (e.g. {10:15:31}+10 adds 10 seconds to the time). Likewise, when adding numbers to dates, the number represents seconds. The result can then be formatting using the default time format by using the time format specifier.

For example:

```
[math]{11:23:45} - {00:00:45}/math]
[math time]{11:23} + (5*60)/math]
```

Equals the following results:

```
45
11:28
```

Note: you may want to convert an integer number to a date or time. Use [Format Days_To_Date] and [Format Seconds_To_Time] to convert integer numbers to their equivalent dates/times. The integer number represents the number of days since midnight, January 1, 0000 and for time it represents the number of seconds since midnight.

```
[Format Days_To_Date]729496[/Format] yields 4/17/1997
[Format Seconds_To_Time]46262[/Format] yields 12:51:02
```

Tip: sometimes you want to calculate something without displaying the results, perhaps while calculating a running total. To do this, put "show=F" into the math parameters, as in $[total=total+[subTotal]]$. This allows you to perform calculations in the middle of a web page without the intermediate numbers appearing to the visitor. Later, you can show the value of the math variable with $total$.

Variables

The $$ context can be used to keep track of variable values while evaluating a page. You may also create up to 150 math variables by name. These named variables can be used in any other $$ context in the template: $fred=12/7.5$...other text here... $fred/15.2$. Note that the name of a math variable is limited to 15 characters in length and must begin with a letter followed by up to 14 letters or numbers. Only the letters a-z and A-Z are allowed.

For example:

```
<![math]found=0[/math]>
...
[founditems]
[showif [name]=John<![math]found=found+1[/math]>[/showif]
[/founditems]
Found [found] of John's Record(s)![/showif]
```

The $$ context in this example is used to add the number of records matching a specified criteria. The first two instances of the $$ context are wrapped in the HTML comment tags to not show the value of the variables in these places. At the end, the value is displayed.

Functions

The $$ context also has built-in functions that can be used in an expression. The following functions are available:

- **Sin** – Returns the sine of a number (degrees).
Example: $\sin(35)$ = .57...
- **Cos** – Returns the cosine of a number (degrees).
Example: $\cos(45)$ = .70...

- . **Tan** – Returns the tangent of a number (degrees).
Example: $\tan(60) = 1.73\dots$
- . **Asin** – Returns the arcsine of a number (degrees).
Example: $\text{asin}(.3) = 17.45\dots$
- . **Acos** – Returns the arccosine of a number (degrees).
Example: $\text{acos}(.6) = 53.13$
- . **Atan** – Returns the arctangent of a number (degrees).
Example: $\text{atan}(.9) = 41.98$
- . **Sinh** – Returns the hyperbolic sine of a number (degrees).
Example: $\text{sinh}(.1) = .10\dots$
- . **Cosh** – Returns the hyperbolic cosine of a number.
Example: $\text{cosh}(.2) = 1.02\dots$
- . **Tanh** – Returns the hyperbolic tangent of a number.
Example: $\text{tanh}(1) = .64\dots$
- . **Exp** – Returns e to the given power.
Example: $\text{exp}(3) = e^3 = 20.07\dots$
- . **Log** – Returns the natural logarithm (ln) of a number.
Example: $\log(5) = \ln 5 = 1.60\dots$
- . **Log10** – Returns the standard logarithm (base 10) of a number.
Example: $\log_{10}(20) = 1.30\dots$
- . **Sqrt** – Returns the square root of a number.
Example: $3 * \text{sqrt}(4) = 6$
- . **Floor** – Rounds a number down to the nearest integer.
Example: $\text{floor}(2.9) = 2$
- . **Ceil** – Rounds a number up to the nearest integer.
Example: $\text{ceil}(2.1) = 3$
- . **Abs** – Returns the absolute value of a number.
Example: $\text{abs}(-2) = 2$
- . **Deg** – Converts a radian value to degrees.
Example: $\text{deg}(\pi) = 180$

- **Rad** – Converts a degree value to radians.
Example: $\text{rad}(45) = .78\dots$

[TIME] TAG

Syntax: [Time]

Placing [Time] in your template displays the current time set on your web server's clock. The format of the time is normally HH:MM:SS (24-hour clock), but you can change the default format by changing the TimeFormat preference. You can also override the default time format by specifying a format inside the tag, such as [Time format=%H:%M:%S]. See [Date] for details.

Text Manipulation

[BOLDWORDS] CONTEXT

Syntax: [boldWords wordlist]Any Text[/boldWords]

Result: Wraps the HTML bold tags around the specified words in the text.

Required Tag Parameter: Wordlist (a comma delimited list of words to bold in the text between the context tags)

To automatically boldface matching words in portions of a template, insert a boldwords context around the text. Any words inside the context that match words in the word list are automatically surrounded by the HTML `` . . . `` tags displaying them as bold in the page.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[BoldWords Grant,John]
John and Grant helped write this product.
[/BoldWords]
```

Anywhere the words Grant or John appear in the contained text, they are wrapped with `Grant` or `John`, respectively. You may use any [xxx] tags in the word list or the container, as in [boldwords [name]]Please embolden this [name][/boldwords]. All [xxx] tags inside the

context are first substituted for their real values, then WebDNA searches for and boldfaces the matching words.

This context is used to assist visitors to more easily view what they are searching for in a database. In the following example, the visitor has previously typed a text string into a search form with an `<input name="wodescriptiondata">` form variable. `[FoundItems]` loops through all of the matching records in the database, and words that match the search text are displayed boldface in the `[description]` field as shown below:

```
[founditems]
Description:
[boldwords [wodescriptiondata]][description][boldwords]
[/founditems]
```

Note: A maximum of 50 words is allowed in the `WordList`. You can put as many words as you like `<i>inside</i>` the context, and all occurrences of matching words will be bolded, but the list of words to look for is limited to 50.

[CAPITALIZE] CONTEXT

Syntax: `[capitalize]some TEXT[/capitalize]`

To convert the words of a sentence to capitalized form, place them inside a `[capitalize]` context. Only the first letter of each word (any letter following a space) is capitalized. If the remaining letters in the word are uppercase, they are automatically changed to lowercase. If the word begins with a numeral, the numeral is left unchanged, and the rest of the letters display as lowercase. No attempt is made to be grammatically correct—articles such as “a, an, of” are capitalized just like any other word.

For example:

```
[Capitalize]Some Text that contains upper- and lower case letters[/Capitalize]
[Capitalize]HI THERE[/Capitalize]
[Capitalize]this is my 1st time at bat[/Capitalize]
[Capitalize]a of on at the in or[/Capitalize]
In the examples above, the displayed text will be:
Some Text That Contains Upper- And Lower Case Letters
Hi There
This Is My 1st Time At Bat
A Of On At The In Or
```

[CONVERTCHARS] CONTEXT

Syntax: [convertchars]Any Text[/convertchars]

Result: Changes 'illegal' characters into legal HTML.

Optional Tag Parameter: db (the conversion database to be used)

To automatically convert certain characters such as the trademark symbol, copyright symbol, or curly-quotes into valid HTML, place the text inside a [ConvertChars] context. The illegal letters are then converted to legal HTML. For example, TM becomes <small>TM</small>, and © becomes ©. This context can be especially useful when your pages contain characters other than those in the standard English alphabet.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[ConvertChars]Some Text that contains  
© or similar letters[/ConvertChars]
```

The above line of WebDNA would produce the following:

```
Some Text that contains &copy; or other unusual letters
```

Anywhere the illegal characters appear inside the context, they are converted to equivalent HTML. You can insert any [xxx] tags inside the context. For example, you can use fields from a database, or even [include] the entire contents of another file. The character conversions are controlled by WebDNA's StandardConversions.db database file.

To add your own conversions, edit the StandardConversions.db file and place the ASCII hex value (using %XX notation) of the character to be converted followed by a tab and the text with which you want to replace that letter (up to 63 characters).

You may optionally specify an alternate database, which contains a list of character conversions:

For example:

```
[ConvertChars db=MyConversions.db]...[/ConvertChars]
```

This provides the ability to perform different conversions, such as multiple ISO character sets in different languages. The database must be of the following form:

```
--- MyConversions.db ---
```

```
from<tab>to
%0B<tab><BR>
%0D<tab><P>
©<tab>&copy;
□<tab><sup><font size=-1>TM</font></sup>
```

Notice you can use escaped-ASCII to specify characters such as Carriage Return (%0D), Soft Return (%0B), Tab (%09), etc. The text in the From column must be only one character, while the text in the To column may be as many as 63 characters.

Note: Normally all database file paths are relative to the local template, or if they begin with "/" they are relative to the web server's virtual host root. You may optionally put "^" in front of the file path to indicate the file can be found in a global root folder called "Globals" inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

[COUNTCHARS] CONTEXT

Syntax: [countchars]Some text[/countchars]

Result: Returns the number of characters (including white space and punctuation) within the context.

For example:

```
[countchars]Some text[/countchars]
```

This example returns 9.

[COUNTWORDS] CONTEXT

Syntax: [countwords delimiters=characters]Some Text[/countwords]

Required Tag Parameter: delimiters (A list of characters used to separate the "words" to be counted. The delimiters may be in URL notation. In order to use the plus sign '+' as a delimiter, place it in a [URL] context or encode it as hex. In URL notation, the plus sign is always converted to a space unless it is in hex notation.)

Result: Counts the number of words inside the context. To count the number of words in something, place the text inside a [CountWords] context.

For example:

```
[CountWords Delimiters= ,]This is a big long sentence, don't you
think?[/CountWords]
```

In the example above, the displayed text is “9.” The number displayed is the number of words inside the context—the number of words found depends on the delimiters specified. If you specify spaces, commas and periods as delimiters, then those characters are not counted and words will be defined as any text that is **not** a delimiter. Long runs of delimiters are ignored, so more than one space between words does not increase the word count.

Note: To count the number of lines in a multi-line string (such as the number of paragraphs in a story), specify carriage return as the delimiter. For example:

```
[CountWords Delimiters=%0D]first line
second line
third line
[/CountWords]
```

This returns “3,” where the “words” are actually defined as whole lines of text. Blank lines do not count. %0D is the hexadecimal equivalent to carriage return.

[CONVERTWORDS] CONTEXT

Syntax: [ConvertWords db=xx.db]Any Text[/ConvertWords]

Result: Changes specified words in a string of text to different words, based on a database of conversions.

To automatically convert certain words into other words, create a database of words to be changed, and put the text to be converted inside a [ConvertWords] context. Any matching words will be changed to corresponding words in the database lookup table.

Example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[ConvertWords db=glossary.db]A HTTP request first uses DNS to look up the ip
address[/ConvertWords]
```

The above line of WebDNA would produce the following (given the proper glossary.db):

```
A Hypertext Transport Protocol request first uses Dynamic Naming System to
look up the Internet Protocol address
```

Here's what the glossary.db file would look like for the example above:

```
-- glossary.db—
from<tab>to
HTTP<tab>Hypertext Transport Protocol
DNS<tab>Dynamic Naming System
ip<tab>Internet Protocol
```

Anywhere the words in the from column appear in the text, they are replaced with whatever is in the to column. There is no limit to the length text in either the from or to columns. You may put any kind of text into either column; for example, HTML is legal in either column.

Parameter	Description
db	(Required) path to conversion database which contains list of "from" and "to" conversions of words to other words
case	(Optional) T or F to indicate that word comparisons should be case-sensitive or not. Default is F, case-insensitive
word	(Optional) SS , WW , SW to indicate that words should be matched as S ub S tring, W hole W ord, or S tart of W ord, same as search parameters when matching text in a database

Some handy uses for [ConvertWords] include removing foul language from online message boards, spelling out acronyms, changing nicknames to full names, inserting hyperlinks, and expanding glossary terms. The following conversion database may give you some ideas:

```
-- conversion.db—
WebDNA Corp.<tab><a href="www.webdna.us">WebDNA Corp.</a>
Mike<tab>Michael
foulword<tab>f*****d
support<tab><a href="mailto:support@webdna.us">support</a>
logo<tab>
```

[DECRYPT] CONTEXT

Syntax: [decrypt seed=8 character seed]Encrypted Text[/decrypt]

Result: Using the same seed value that was used to encrypt a block of text, [decrypt] will decrypt the text. Without the proper seed, the text cannot be decrypted.

Required Tag Parameter: seed (The same 8 character seed used to encrypt the block of text. **Note:** This parameter is not required for method=base64 decrypting.)

Optional Tag Parameter: method (Either “CyberCash” or “Base64”. If not specified, then standard WebDNA decryption is assumed. CyberCash is the triple-DES encryption used to communicate with the CyberCash Cash Register servers. Base64 is the encoding (not safe for encryption) that standard HTML browsers use for Basic Authentication. You can store sensitive data, like credit card numbers in a database as encrypted text. A special password protected page can let you view the original credit card number.)

For example:

```
[decrypt seed=ABcd12#$][EncryptedCCNumber]/[decrypt]
```

Note: You must supply a seed value to WebDNA’s [decrypt] context. For safety, all data that is encrypted using WebDNA’s internal seed (no seed value specified to the [encrypt] context) may not be decrypted.

[ENCRYPT] CONTEXT

Syntax: [encrypt seed=8 character seed]Any Text[/encrypt]

Optional Tag Parameters:

- seed (Any 8-character seed that can be used to [decrypt] the text. For CyberCash, this should be the MerchantKey you were assigned when you created a CyberCash merchant account. For standard WebDNA encryption, this is your secret key for decryption later. CyberCash encryption is one-way; it cannot be decrypted by your server. This parameter is not required for method=base64 encryption.)

- `method` (Either “CyberCash” or “Base64”. If not specified, then standard WebDNA encryption is assumed. CyberCash is the triple-DES encryption used to communicate with the CyberCash CashRegister servers. Base64 is the encoding (not safe for encryption) used by standard HTML browsers for Basic Authentication. Do not attempt to encrypt more than 48 bytes using `method=Base64`.)
- `file` (Specifies a file that is to be encoded using Base64. This is useful for sending e-mail attachments using the WebDNA `sendmail` context. Note that anything between the opening and closing `encrypt` tag will be ignored if this parameter is present.

Note: This only applies to WebDNA version 4.0.2rc1 and above.)

- `emailformat` (For Base64 only, this specifies if the resulting encoded string should contain line breaks suitable for e-mail applications. Valid values are either 'T' or 'F' the later being the default. This should be used in conjunction with the `file` parameter when sending e-mail attachments from a WebDNA template.

Note: This only applies to WebDNA version 4.0.2rc1 and above.)

Result: Using a specified seed value, the `[encrypt]` context encrypts the enclosed block of text. The 8-character seed may be any combination of letters, numbers or other characters.

Use `[Decrypt]` to decode the text. To decrypt a sequence of text, you must set the seed value to exactly the same as when you encrypted it. The seed may be a sequence of up to 8 characters (anything you can type on the keyboard, except '&' or '=' or '[' or ']'). Do not ever let anyone know what the seed value is, because that would allow him or her to decrypt the text.

This context is most often used to store passwords in a database (such as WebDNA's own `Users.db`), so that in the unlikely event that someone is able to download the file, the passwords will be unreadable.

Caution: Do not lose your seed value: once the text is encrypted, the same seed value must be supplied to the `[decrypt]` context to return the text to normal. This is because the `[encrypt]` context works in such a way that the same text, using the same seed, does not encrypt the same way every time. Thus you cannot simply compare the encrypted text to see if the original text is the same.

Note: Do not ever forget your passwords or seed values! Even WebDNA Software Corp. cannot decrypt something once it has been encrypted!

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[encrypt seed=abCD12%&]SomeText[/encrypt]
```

However, there are certain situations where the encrypted text should always be the same for the same source text. The passwords stored in the Users.db file are this way so they don't have to be decrypted in order to compare them. If you [encrypt] text without supplying a custom seed value, WebDNA uses an internal seed to encrypt the text ensuring the same source text always creates the same encrypted text.

For example:

```
[encrypt]Password[/encrypt]
```

Note: For safety, text encrypted without a seed value may NOT be decrypted.

You may also encrypt entire WebDNA templates. See *Encrypting Templates* for more information.

[FORMAT] CONTEXT

Syntax: [Format FormatSpec]Text or Number[/Format]

Formats text or numbers in various widths or currency formats.

To display numbers with various decimal points or currency formats, surround the number or text with a [Format] context.

For example, normally you would put the following text into a .tpl file on your server and use a web browser to link to it:

```
[Format 10.2f]99.5[/Format] (f stands for floating-point number)
```

```
[Format 10s]Hello[/Format] (s stands for string of text)
```

```
[Format Days_To_Date %m/%d/%y]195462[/Format]
```

```
[Format Seconds_To_Time]49768[/Format]
```

```
[Format Seconds_To_Time %!:%M:%S %p]49768[/Format]
```

```
[Format thousands 14.2f]394363210[/Format]
```

```
[Format thousands 14,2f]394363210[/Format]
```

```
[Format thousands .3d]7[/Format] (d stands for decimal number)
```

The number displays right justified with enough preceding spaces and digits after the decimal point to fill the exact width of the format specifier. Text is left justified, with enough spaces after it to exactly fill the width specifier.

| 99.50| (10 wide, 2 after the decimal)
|Hello | (10 wide, text)
|13:49:28|
|01:49:28 PM|
|04/07/1997| (#days as a date)
|394,363,201.00| (14 wide, number with thousands separator)
|394.363.201,00| (14 wide, number with European thousands separator)
|007| (3 wide, integer part of number only, zeroes preceding)

Given a number 345.67, the following format specifiers will display as shown:

8.3f = | 345.670| (f stands for floating point)
8.2f = | 345.67|
8.1f = | 345.7| (notice rounding from .67 to .7)
8.0f = | 346| (notice rounding from .67 to next higher integer)
5d = |00345| (notice no rounding, and preceding 0s to fill 5 digits)

Optional date format—to format a number as a date (the number must represent the number of days since Midnight, January 1st, 0000), use the optional date specifier and a date format, the same as from the [date] tag. Also see date and time [Math].

[GETCHARS] CONTEXT

Syntax: [getchars start=1&end=3]the text[/getchars]

Result: Returns just the characters, including the starting and ending index values, from the text within the context.

Required Tag Parameters: Start (the starting index (the first characters index is 1) to start getting characters from)

Optional Tag Parameters:

- **End** – The ending index of characters to return. If no end parameter is used, the last character in the context is considered the end value.
- **from**– Possible value is “end”. This allows you to get characters from the end of the text. In this case, the first index value, 1, is the last character within the context.
- **Trim** - to remove leading and trailing white space in a given text string.

To display a subsection of some text, put the text inside a [getchars] context. Most often, the contents of the [getchars] context will be the value of an input field or database field that WebDNA will replace.

Example 1:

```
[getchars start=1&end=2][field1][/getchars]
```

Example 2:

```
[GetChars start=14&end=18]Hello there, Edwina[/GetChars]
```

Example 2 returns "Edwin", which is the sequence of letters starting at the 14th position and extending to the 18th position. Extracting sub-portions of text is useful when you need to get text from files.

Example 3:

```
[GetChars start=1&end=2&from=end]Hello there, Edwina[/GetChars]
```

Example 3 returns "na", which is the sequence of letters starting at the 1st position from the end and extending to the 2nd position from the end.

You can use the new '**TRIM=Right/Left/Both**' parameter for the [GetChars] context to remove leading and trailing white space in a given text string.

Example WebDNA:

```
[text]test_string=  
    This is a test.  
[/text]
```

Example input string: "[test_string]"

Remove leading white space:

```
>[getchars start=1&end=&trim=left][test_string][/getchars]<
```

Remove trailing white space:

```
>[getchars start=1&end=&trim=right][test_string][/getchars]<
```

Remove leading and trailing white space:

```
>[getchars start=1&end=&trim=both][test_string][/getchars]<
```

Remove leading and trailing white space, and retrieve characters in the 1-4 positions:

```
>[getchars start=1&end=4&trim=both][test_string][getchars]<
```

Remove leading and trailing white space, and retrieve characters in the 1-5 positions, from the end:

```
>[getchars  
  start=1&end=5&trim=both&from=end][test_string][getchars  
]<
```

Results...

Example input string: "

 This is a test.

"

Remove leading white space:

```
>This is a test.
```

```
<
```

Remove trailing white space:

```
>
```

 This is a test.<

Remove leading and trailing white space:

```
>This is a test.<
```

Remove leading and trailing white space, and retrieve characters in the 1-4 positions:

```
>This<
```

Remove leading and trailing white space, and retrieve characters in the 1-5 positions, from the end:

```
>test.<
```

Note that you still need to supply the 'start' and 'end' parameters.

[GREG] CONTEXT

Syntax: [[Grep search=regexp&replace=regexp]Any Text[/Grep]

Result: Replaces text based on a regular expression. This popular UNIX utility has been adapted to WebDNA.

Example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[Grep search=([0-9]*-[0-9]*)&replace=<b>\1</b>]Hi, my phone number is 555-1234, and I'd like you to call me[/Grep]
```

In the example above, the displayed text will be:

Hi, my phone number is 555-1234, and I'd like you to call me

Parameter	Description
search	(Required) Regular expression that defines what text to search for in the body of the context
replace	(Required) Regular expression that defines how to output the resulting text
New for 5.0 IgnoreCase	(Optional) ignores case sensitivity while performing the grep function

There is a new 'IgnoreCase' parameter for the [GREG] context. Here is an example:

Search for 'usa' and replace with 'USA'

```
[grep search=usa&replace=USA&ignorecase=T]
I was born in the usA

I was born in the uSa

I was born in the Usa

[/grep]
```

Other examples...

```
[grep search=[0-9]*(ScOtT) [0-9]*&replace=___\1___&IgnoreCase=T]123scott321
[/grep]
[grep search=staging&replace=development&IgnoreCase=T]/Staging
/myfile[/grep]
```

Results:

```
I was born in the USA
I was born in the USA
I was born in the USA
```

Other examples...

```
___scott___
/development/myfile
```

[INPUT] CONTEXT

Syntax: [input]Any Text[/input]

Result: Prepares text for use in input forms using <textarea> tags.

To prepare text fields that may contain carriage returns for use in <textarea> input fields in a form, place the text inside an [Input] context. Certain characters, such as carriage returns, are converted to “soft-returns” when they are saved in a database file. The [Input] context converts them back to “hard-returns” so they look right when displayed inside <textarea> tags.

For example:

`<textarea>[input][fieldName][input]</textarea>`

This context is most often used when you create a form that lets visitors edit large portions of text in a database field that has carriage returns in it. If you don't use this context, then the lines of text inside the `<textarea>` multi-line input field appear to have "lost" all the carriage returns originally entered. This is because web browsers don't understand how to display a "soft-return" character. Instead, they display it as a space. Additionally, HTML text in the edit area can be confused with HTML in the enclosing page. As such the `[input]` context converts all angle brackets to `>` and `<` to display them properly.

The following list explains what happens to text placed inside an `[Input]` context:

If the letter is...	then it is translated to...
Soft-return (%0B)	Hard-return (%0D)
<	<
>	>
&	&
"	"
Any other letter	no change

Note: There may not be a reason to use this context unless it's inside `<textarea>`, as no other input field allows carriage returns typed in it.

[LISTPATH] CONTEXT

Syntax: `[listpath path=folder1/file.html][name][/listpath]`

Result: Allows you to specify the specific components of a file path. This context loops through all components.

Required Tag Parameters: `path` (URL-style path to be broken up)

Optional Tag Parameters:

- **fileOnly** – If the value of this parameter is "T" (True), then only the last component of the path is returned. If the path ends in a slash, then the filename is considered to be blank "".
- **pathOnly** – If the value of this parameter is "T" (True), then the components leading up to the filename are listed.

Optional Context Tags:

- **[index]** – From 1 to the number of components looped through. If the path ends in a slash, a final (extra) blank component for the file name is added: "".
- **[name]** – The value for each component being listed. This may be a folder or file name.
- **[filename]** – This value is always available and can be used to decide whether to loop through a path (in which case [FileName] is blank "") or a file path.

To display a list of all the separate folder names in a path (text separated by "/"), use a [ListPath] context. You may optionally specify that only the filename will be displayed (subtracting any path leading to it), or only the folder names (subtracting the filename from the end).

ListPath is only used for text-manipulation – it does not actually look at your hard disk. To find all the files in a folder on your hard disk, use [ListFiles] instead.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[ListPath path=/folder1/folder2/folder3/filename.txt]
  Path part #[index]: [name]<br>
[/ListPath]
```

The example above yields:

```
Path part #1: folder1
Path part #2: folder2
Path part #3: folder3
Path part #4: filename.txt
```

```
[ListPath path=/folder1/folder2/folder3/filename.txt&FileOnly=T]
  Path part #[index]: [name]<br>
[/ListPath]
```

The example above yields:

```
Path part #1: filename.txt
```

```
[ListPath path=/folder1/folder2/folder3/filename.txt&PathOnly=T]
  Path part #[index]: [name]<br>
[/ListPath]
```

The example above yields:

```
Path part #1: folder1
```

Path part #2: folder2
Path part #3: folder3

[LISTWORDS] CONTEXT

Syntax: [listwords words=text&delimiters=characters]Text[/listwords]

Result: Displays a list of all the separate words in a string. You can also specify delimiters to define the boundaries of words (usually spaces, commas and periods).

Required Tag Parameters: words=text (the text that you would like to break up into individual words)

Optional Tag Parameters:

- **delimiters=characters** – List of single characters defining word boundaries. Defaults to space and comma. The delimiters can be specified URL-style (as hex values after the percent sign) using the [URL] context. In particular, you can use the plus sign as a delimiter if you enclose it in the [URL] context since a plus is considered a space in a URL.
- **tabs=T** – Setting Tabs=T causes the list of words to break at tab boundaries only, and runs of tabs are not collapsed. This assists in parsing special formats where two tabs in a row are important and should not be skipped.

Optional Context Parameters:

- **[index]** – A number from 1 to the number of words in the text.
- **[word]** – The word separated from the text.

Example 1:

```
[ListWords Words=This is a big long sentence]
[index]: [word]<br>
[/ListWords]
```

The example above yields the following:

```
1: This
2: is
3: a
4: big
5: long
6: sentence
```

Example 2:

```
[ListWords Words=Hello.    My name is Carl!&Delimiters= ,!]  
[index]: [word]<br>  
[/ListWords]
```

The example above yields the following. The “.” at the end of “Hello.” is shown as part of the word, because “.” was not specified in the list of delimiters. Also note the long run of spaces in a row is collapsed and does not appear in the word list:

```
1: Hello.  
2: My  
3: name  
4: is  
5: Carl
```

[LOWERCASE] CONTEXT

Syntax: [Lowercase charset]Any Text[/Lowercase]

Result: Changes all upper case letters to lower case.

To convert certain characters to lower case, put them inside a [Lowercase] context.

For example, normally you would put the following text into a .tpl file on your server and use a web browser to link to it:

```
[Lowercase]Some Text that contains upper- and lower case letters[/Lowercase]
```

In the example above, the displayed text will be some text that contains upper- and lowercase letters. This context is included for completeness as a complement to [Uppercase].

Note: you can optionally change the behavior of lowercase with charset=mac (common on Macintosh) or charset=iso (common on PCs), which causes high-ASCII characters to change case differently depending on which character set your data is.

```
[Lowercase charset=iso]...some text...[/Lowercase]
```

[MIDDLE] CONTEXT

Syntax: [middle startafter=text&endbefore=text]Some text[/middle]

Result: To display a subsection of some text, put the text inside a [Middle] context.

Required Context Tags:

- **StartAfter=text** – A string of text characters to search for that define the beginning of the text to be returned. All preceding text (and the StartAfter text itself) are ignored.
- **EndBefore=text** – A string of text characters to search for that define the end of the text to be returned. All following text (and the EndBefore text itself) are ignored.

For example:

```
[Middle StartAfter=<body>&EndBefore=</body>]
<html>
<head>
</head>
<body>
Hi There
</body>
</html>
[/Middle]
```

The example above returns “Hi There”, which is the sequence of letters between “<body>” and “</body>”. Extracting sub-portions of text like this is useful when removing the HTML header information from a file (or web page) containing HTML that you want to include inside the body of a web page.

[RAW] CONTEXT

Syntax: [raw]Any Text, including [xxx][/raw]

Result: Displays enclosed text without interpreting the [xxx] tags in any way.

To prevent WebDNA from interpreting [xxx] tags, enclose the text inside a [Raw] context. The text inside the container is displayed unchanged until the [/raw] tag is found.

For example:

```
[raw]Any text, including any [xxx]
tags such as [date] or [delete][/ raw]
```

In the example above, the displayed text equals:

```
Any text, including any [xxx] tags such as [date] or [delete].
```

This context is most often used within WebDNA's documentation to make it possible to display the unmodified tags without them being first interpreted. Also, any HTML pages on your site that have many [or] characters might be misinterpreted by WebDNA, so by surrounding those pages with <![raw]>...entire text of page...<![/Raw]>. Notice the <![> HTML comments in this example: they are used so that if this page is ever displayed without being processed by WebDNA, the[Raw] tags will be invisible.

RAW COMMAND

Syntax: Raw

Result: Returns the "raw" contents of a template.

Sometimes you, or someone from technical support, needs to view the text of a template file in its raw form without being processed by WebDNA. Usually this is so you can see the WebDNA commands rather than the final resulting HTML those WebDNA commands generate. WebDNA normally intercepts URLs leading to template files and makes it impossible to "view the source" of a WebDNA template.

The Raw command is provided for sites using Suffix Mapping to automatically process all files ending in a particular extension. In this scenario, it is impossible to link to the file directly to view the raw contents of the file. While this is good because it does not allow the outside world to see the contents of your pages, it can make site development difficult if you are debugging a particular page remotely. The Raw command should be protected with the Admin password so it is not available to everyone accessing your site.

[REMOVEHTML] CONTEXT

Syntax: [RemoveHTML]Any Text[/RemoveHTML]

Result: Removes HTML or WebDNA tags from a string of text.

To automatically strip out HTML or WebDNA tags from text, put the text inside a [RemoveHTML] context. Any HTML found in the text will be removed, leaving just the plain non-HTML text behind. Alternately you can specify that WebDNA be removed instead.

For example, normally you would put the following text into a .tpl file on your server and use a web browser to link to it:

```
[RemoveHTML]Some Text that contains <br> <h3>or</h3> other  
HTML[/RemoveHTML]
```

The above line of WebDNA would produce the following:

```
Some Text that contains or other HTML
```

Anywhere HTML sequences appear inside the context, they will be removed. You may put any [xxx] tags inside the context—for instance, you may use fields from a database, or even [include] the entire contents of another file. HTML is recognized as any text that begins with “<” followed by any number of non-space characters, followed by any number of characters, followed by “>”.

Parameter	Description
RemoveWebDNA	(Optional) T or F to indicate that you also want to remove WebDNA from the text. HTML is always removed from the text. Default is to remove HTML only.
ReplaceWith	Text with which you want to replace the HTML sequences. Defaults to nothing.

[TEXT] CONTEXT

Syntax: [Text]Text Variable Assignment[/text]

Result: Stores a text variable that can be used later in the template.

To remember some text for use later in a page, assign a text variable using the [Text] context. You may assign as many named text variables (similar to [math] variables, except they store textual information instead of numbers) as you want per page. The length of the text is also only limited to RAM.

Optional Tag Parameters:

- **multi=Boolean** – “T” or “F”. Allows you to assign more than one text variable in a single context.
- **show=Boolean** – “T” or “F”. Default behavior is to hide text when assigning to a text variable. If you want the text to be shown at the same time it is assigned to a variable, you may set Show=T.

Optional Context Parameters:

- **[variable]** – Inserting the name of any variable that you create with the [text] context in square brackets will be replaced by the value of that variable.

For example: (normally you would put the following text into a .tpl file on your server and used a web browser to link to it)

```
[Text Show=F]sentence1=This is a big long sentence, don't you think?[/Text]
[Text]name=John[/Text]
[Text]sentence1[/Text]
[Text]name[/Text]
[name]
```

In the example above, the following text displays:

```
This is a big long sentence, don't you think?
John
John
```

The Text context looks for an equal sign “=” in the body of the text, and if it finds one then it assumes you are assigning a new value to a new text variable. If no equal sign is present in the body of the text, then it assumes you want to retrieve the value of a previously stored variable. Reassigning a text variable to a new value replaces the previous value.

You do not have to use the [Text] context to display a text variable; you may simply put the variable name inside brackets like so: [varname]. Both [Text]varname[/Text] and [varname] will display the same thing.

Parameters	Description
------------	-------------

Parameters	Description
multi	(optional) "T" or "F". Allows you to assign more than one text variable in a single context. [Text multi=T]var1=Joe&var2=Fred[/Text] simultaneously assigns two variables, named "var1" and "var2" to the values "Joe" and "Fred."
secure	(optional) "T" or "F". Defaults to "T". Setting secure=F this text variable can be overridden by incoming form variables. This is not recommended, and the default behavior is secure. If you really want outside visitors to your web site to be able to change the values of your internal text variables (usually with a <form>), then you may use the non-secure version of this parameter.
show	(optional) "T" or "F". Default behavior is to hide text when assigning to a text variable. If you want the text to be shown at the same time it is assigned to a variable, you may set Show=T. If you set multi=T at the same time, then no text will be displayed.

[UnURL] CONTEXT

Syntax: [unurl]URL text[/unurl]

Result: To automatically convert text containing URL characters such as %20, %3A, etc, place it inside a [UnURL] context. Certain letters such as spaces, colons, and the equals sign (=) are not allowed inside URLs unless they are first converted to hexadecimal form—the UnURL context converts them back. This is the opposite of [URL].

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[UnURL]Filename%20with%20spaces.gif[/UnURL]
```

In the example above, the text displays as:

```
Filename with spaces.gif
```

Note: By default, the [UnURL] context will not convert hex codes that contain lowercase letters. So, %3A will be converted to a colon while %3a will not. Starting with WebDNA version 402rc1, you can use the 'IGNORECASE' parameter to force the [UnURL] context to convert hex codes containing

lowercase letters. So `[UnURL IGNORECASE=T]%3a[/UnURL]` will convert `%3a` to a colon.

This context is rarely needed, because most of the time WebDNA has already converted the text (in URL parameters, for instance) back to plain text. If you plan to store text with embedded unusual characters such as tabs or carriage returns into a field of a database, you might use `[URL]` to store them and `[UnURL]` to retrieve them.

[URL] CONTEXT

Syntax: `[url]Any Text[/url]`

Result: Changes text to URL-compatible characters.

To automatically convert text containing spaces or other “illegal” URL characters, place it inside a `[url]` context. Certain letters, such as spaces, colons, and the equals sign (=) are not allowed inside URLs unless they are first converted to hexadecimal form.

For example:

```

```

In the example above, the text displays as:

```

```

This context is frequently used with `[sku]` fields that represent the name of a picture of a product, as in `` because many times a product sku will have “illegal” characters in it.

See the discussion on *Important Topics* to determine when to use the `[url]` context.

[UPPERCASE] CONTEXT

Syntax: `[uppercase]Any Text[/uppercase]`

Result: Changes all lower case letters to upper case.

Optional Tag Parameter: `charset` (Either “iso” or “mac” depending on how you want high ASCII characters (decimal value greater than 127) treated.)

To convert certain characters to upper case, place them inside an [uppercase] context.

For example:

```
[uppercase]Some Text that contains upper- and  
lower-case letters[/uppercase]
```

In the example above, the text displays as:

```
SOME TEXT THAT CONTAINS UPPER- AND LOWER-CASE LETTERS
```

This context is most often used with [UserName] and [Password] tags, because many web servers do not allow lowercase letters to be sent from the browser's realm protection dialog. By making sure that all usernames / passwords entered into a database are uppercase, you can be sure that all browsers and web servers will work properly for password authentication. The [UpperCase] . . . [/UpperCase] context is also useful to force fields to be consistent when the data will be used with the [LookUp] command as with the password database. The consistency improves the speed with which the data is found.

For example:

```
[uppercase charset=iso]å[/uppercase]
```

This displays "Å" if å has the value specified in the ISO-Latin1 character set.

Passwords

[AUTHENTICATE] TAG

Syntax: [authenticate *Some Text*]

Placing [Authenticate] in your template causes the remote browser to display the Username/Password dialog with whatever text you specify. When the visitor types a username and password into their browser, your templates can get that information by including the [Username] and [Password] tags. Normally you [Authenticate] is placed inside [ShowIf] or [HideIf] contexts comparing the username/password, otherwise the password dialog continues popping up forever.

Note: [Authenticate] is a low-level tool requiring some sophisticated knowledge to use, so most people use the simpler [Protect] tag to password-protect a page. You can see an example of how [Protect] makes use of

[Authenticate] by reading the WebDNA in the file “MultiGroupChecker” on your hard disk (in the WebCatalogEngine folder).

[PASSWORD] TAG

Syntax: [Password]

Result: Placing [Password] in your template displays the password entered into the remote browser’s last Realm password dialog. See [UserName] and [Authenticate].

[PROTECT] TAG

Syntax: [Protect *group1,group2*]

Result: Placing [Protect Groups] in your template causes the remote browser to display the Username/Password dialog until the visitor enters a username/password belonging to one of the specified groups. The Users.db database provided with WebDNA contains all the username/password/group information.

[USERNAME] TAG

Syntax: [UserName]

Result: Placing [UserName] in your template will display the username entered into the remote browser’s last Realm password dialog. See [Password] and [Authenticate].

Files and Folders

[APPENDFILE] CONTEXT

Syntax: [AppendFile FileName]Text[/AppendFile]

Result: Writes text to the end of an existing file.

To add text to the end of an arbitrary text file, place an [AppendFile] context into a template. [AppendFile] creates a new file if one does not already exist.

All text is placed at the end of the file. The file must not be a database file currently open and in use by WebDNA. See [WriteFile] for further details.

Note: [AppendFile] does not ‘understand’ databases. To append a new record to the end of a database, use Append instead.

For Example: Usually, the following text is placed into a .tpl file on your server then uses a web browser to link to it):

```
[AppendFile SomeTextFile]Hello, my name is Grant. The time is [time]
This is a second line[/AppendFile]
```

The text file “SomeTextFile” opens, and displays the following text:

```
Hello, my name is Grant. The time is 13:43:01
```

A second line is written at the end of the file. Notice that carriage returns inside the context are written to the file exactly as they appear. Also notice that any WebDNA [xxx] tags inside the context are substituted for their real values before being written to the file. You may specify a full or partial path to the file, as in “/Some Folder/file.txt” (starting from the web server’s root) or “LocalFolder/file.txt” (starting in the same folder as the template file, looking down into a folder called “LocalFolder”).

Security Note: By default, all files created by WebDNA are tagged with a special code telling WebSTAR not to display them via URL. If you want files to be visible to outside browsers, use the optional settings below.

Parameter	Description
Secure	“T” for files that should be secure—WebSTAR will not display them. “F” for files that should be visible via URL—WebSTAR will display them. For example: [WriteFile secure=F&file=SomeFile]...[/WriteFile]
File	When you use the secure option above, you must also provide the name (or relative path) of the file to create.

[CALCFILECRC32] TAG

New for 5.0

Syntax: [CalcFileCRC32 file=...]

Calculates the CRC32 value of a given file.

Example:

```
[CalcFileCRC32 file=../header.inc]
```

Results:

```
40166762401
```

[COPYFILE] TAG

Syntax: [CopyFile from=FromFile&to=ToFile]

Placing [CopyFile from=fred&to=wilma] in your template immediately copies the file called “fred” to another file called “wilma.”

[COPYFOLDER] TAG

Syntax: [CopyFolder from=FromFolder&to=ToFolder]

Result: Placing [CopyFolder from=FromFolder&to=ToFolder] in your template immediately copies the folder called “fredFolder” to another folder called “wilmaFolder.” The original folder remains where it is currently.

[CREATEFOLDER] TAG

Syntax: [CreateFolder path=folder]

Result: Placing [CreateFolder StarShip/Troopers] in your template immediately creates a new folder with the name “Troopers” inside a folder called “StarShip” which is in the same folder as the template itself.

[DELETEFILE] TAG

Syntax: [DeleteFile file=FilePath]

Result: Placing [DeleteFile file=fred] in your template immediately deletes the file called “fred” in the same folder as the template. Paths are relative to the template, so “somefolder/fred” deletes the file down a level from the template inside a folder called “somefolder,” while “../fred” deletes the file in the folder one level up from the template.

[DELETEFOLDER] TAG

Syntax: [DeleteFolder path=FolderPath]

Result: Placing [DeleteFolder path=StarShip] in your template immediately deletes the folder called “StarShip” in the same folder as the template. Paths are relative to the template, so “somefolder/StarShip” deletes the folder down a level from the template inside a folder called “somefolder,” while “../StarShip” deletes the folder inside the folder one level up from the template. **All files inside the folder are deleted, and all subfolders are deleted, as though you had dragged the folder to the trashcan and emptied it immediately.**

Warning: You can not undo this operation!

Note: if the parameter to [DeleteFolder] is an alias, then just the alias itself is deleted.

[FILECOMPARE] TAG

New for 5.0

Syntax: [filecompare params...]

The file compare tag compares the size, date, or CRC32 value of two given files.

Optional Tag Parameters:

- **method** - Size | Date | CRC32
- **file1** - path of first file.
- **file2** - path of second file.
- **file1crc** - arbitrary crc32 value, in place of an actual path for file1. (useful if you want to store the CRC32 value of a file and use it at a later time to test if the file contents have been changed)

results: method=Size

SMALLER - file2 is smaller than file1
LARGER - file2 is larger than file1
SAME - file1 and file2 are the same size

results: method=Date

OLDER - file2 is older than file1
NEWER - file2 is newer than file1
SAME - file1 and file2 have the same create date

results: method=CRC32

DIFFERS - the files differ in content
SAME - the files have the same content

Errors:

ERROR_FILE1 - 'file1' could not be found or opened
ERROR_FILE2 - 'file2' could not be found or opened
ERROR_BOTH - neither file could be found or opened

Examples:

```
[filecompare  
method=SIZE&file1=../create_tutorial.tpl&file2=../header  
.inc]
```

```
[filecompare  
method=DATE&file1=../create_tutorial.tpl&file2=../header  
.inc]
```

```
[filecompare  
method=CRC32&file1=../create_tutorial.tpl&file2=../heade  
r.inc]
```

```
[text]crc_value=[CalcFileCRC32 file=../header.inc][text]  
[filecompare  
method=CRC32&file1crc=[crc_value]&file2=../header.inc]
```

Results:

```
LARGER  
OLDER  
DIFFERS  
SAME
```

[FILEINFO] CONTEXT

Syntax: [fileinfo file=filepath]various file information[/fileinfo]

Result: Allows you to get various bits of information about a particular file.

Required Tag Parameter: <no name> (URL-style path to the file)

Optional Context Tags:

- **[isFolder]** – “T” or “F” depending upon whether the path leads to folder.
- **[isFile]** – “T” or “F” depending upon whether the path leads to a file.
- **[exists]** – “T” or “F” depending upon whether the path leads to a file that exists on the drive.
- **[fileName]** – The name (last component) of the path specified.
- **[fullPath]** – A platform-specific path to the file that may be a full path or partial path.
- **[createdate]** – The date the file was created.
- **[createtime]** – The time the file was created.
- **[modDate]** – Last date the file was modified.
- **[modTime]** – Last time the file was modified.
- **[size]** The size of the file in bytes.
- **[startPath]** – The path (minus the file name).

The [fileinfo] context allows you to quickly get information on a particular file without having to use the [listfiles] context to loop through all the files in a folder.

For example:

```
FileInfo ../Pictures/1234.gif  
File Name: [FileName]<br>  
Modified: [ModDate]<br>  
Size: [Size]<br>  
[/FileInfo]
```

The [fileinfo] context is very useful to deciding whether to include an image that may or may not exist. This helps prevent “broken” images. However, since it is slower than just including the image, it should only be used when existence is in doubt.

For example:

```
[showif [fileinfo images/test.gif][exists][fileinfo]=T]

[/showif]
```

[LISTFILES] CONTEXT

Syntax: [Listfiles *path*= *FolderPath*]File Tags[/Listfiles]

Result: Lists all the files in the specified folder.

Parameter	Description
path	(Required) The path to the folder that is to be listed. This path is relative to the template
ShowInvisibles	(Optional) Macintosh only: if set to T , then display invisible filenames, if set to F , do not display invisible filenames
New for 5.0 Name	(Optional) filters files by name
New for 5.0 Exact	(Optional) if set to T , then display filenames that match 'Name' criteria, if set to F , exact match of filenames that match 'Name' criteria not required

Optional Context Tags:

- **[Isfolder]** – “T” if this item is actually a folder. “F” otherwise.
- **[Isfile]** – “T” if this item is actually a file. “F” otherwise.
- **[FileName]** – Name of the file.
- **[Fullpath]** – Full path to the file, including its containing FolderPath.
- **[Createdate]** – Date the file was originally created.
- **[Createtime]** – Time the file was originally created.
- **[Moddate]** – Last date the file was modified.
- **[Modtime]** – Last time the file was modified.
- **[Size]** – The file’s size, in bytes.

- **[Index]** – A number from 1 to the number of files indicating this file's position in the list.
- **[Startpath]** – The folder path leading to the file.

To display a list of all the files in a particular folder, insert the [ListFiles] context into a WebDNA template. An absolute or relative path to the folder may be specified just as you would specify an absolute or relative URL. The folder name may be a WebDNA [xxx] tag. The context tags available vary depending on the platform used as some file information on a hard drive is platform specific.

For example:

```
[Listfiles path=../WebDNA/
File Name: [FileName]<br>
Modified: [ModDate]<br>
Size: [Size]<br>
[/Listfiles]
```

You can use the 'Name' and 'Exact' parameters with the [listfiles] context to 'filter' the results.

For example:

```
[listfiles path=../&name=tpl&exact=f]
[filename]

[/listfiles]
```

Results:

```
create_tutorial.tpl
```

[MOVEFILE] TAG

Syntax: [MoveFile from=FromFile&to=ToFile]

Placing [MoveFile from=fred&to=wilma] in your template immediately moves the file called "fred" to another file called "wilma," and deletes the file "fred" after the move is complete. You may specify a different folder within which to move the file.

[RENAMEFILE] TAG

Syntax: [RenameFile from=oldname&to=newname]

Result: Placing [RenameFile from=fred&to=wilma] in your template immediately renames the file called “fred” to “wilma.”

[WAITFORFILE] CONTEXT

Syntax: [waitforfile file=filepath]Text[/waitforfile]

Result: To perform an action as soon as a file appears on disk, put WebDNA inside of a [WaitForFile] context. The server will wait until the specified file appears on disk, then execute the text/WebDNA inside the context.

Required Tag Parameter: file=filepath (URL-style path to file to create and/or write over.)

Optional Tag Parameter: timeout=seconds (Number of seconds to wait before canceling the wait. If the timeout value is met the text in the context is not displayed (or executed).)

Note: Unless you specify a timeout, WaitForFile will wait forever until the file appears. The template will not finish executing and the HTML will not be returned to the visiting browser until the WaitForFile finishes.

For example:

```
[WaitForFile file=ICVer001.ans]
  Some WebDNA to parse the ICVerify answer file format and do something
  important with it
[/WaitForFile]
```

[WRITEFILE] CONTEXT

Syntax: [writefile file=filepath&secure=boolean]Text[/writeFile]

Result: Creates a file, if necessary, and writes text to the beginning of a file.

Required Tag Parameter: file=filepath (URL-style path to file to create and/or write over.)

Optional Tag Parameter: secure=Boolean (“T” or “F” depending upon whether the file should be visible to a remote browser.)

See [AppendFile] for description of secure.

If you are using the [writefile] . . . [/writefile] context as a method for creating dynamic HTML, you MUST explicitly state [writefile file=filepath&secure=F]

Note: WriteFile does not ‘understand’ databases. If you want to write a new record to the end of a database, use Append instead.

For example:

```
[WriteFile SomeTextFile]
Hello, my name is Grant. The time is [time]
This is a second line[/WriteFile]
The text file "SomeTextFile" is created if necessary and opened, and the text
Hello, my name is Grant. The time is 13:43:01
This is a second line written to the file.
```

Notice that carriage returns inside the context are written to the file exactly as they appear. Also notice that any WebDNA [xxx] tags inside the context are substituted for their real values before being written to the file.

Technical

[APPLESCRIPT] CONTEXT

Syntax: [AppleScript]AppleScript Program[/AppleScript]

Result: Executes the AppleScript contained in the context and displays the results.

To embed the results of an AppleScript into a page, insert an AppleScript context into the template. The AppleScript program contained inside the context is executed, and any returned value is displayed in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the AppleScript is executed.

For example:

```
[AppleScript]if [random]>50
return "It's more than 50!"
else
return "It's less than 50!"
end if
[/AppleScript]
```

In this example, the tag [random] is first substituted with a random number, then the resulting AppleScript is executed. Any WebDNA [xxx] tags are allowed inside the AppleScript program, even if they represent complex lookups or searches. The WebDNA tags are always evaluated first, and the resulting text is then executed as an AppleScript program.

AppleScripts have complete access to your hard disk and all programs on your web server. You can write an AppleScript that erases the contents of your hard disk. Don't worry about remote visitors to your web site executing their own AppleScripts remotely. They can only execute an AppleScript inside a template file you have saved on your web server's hard disk.

[COMMAND] TAG

Syntax: [Command]

Result: Placing [Command] in your template displays the WebDNA command that was used to get to this page. Some examples are ShowPage, Search, ShowCart, etc. These commands are often used in Form Actions, but can also be specified in an HREF hyperlink.

[DDECONNECT] CONTEXT

Syntax: [ddeconnect program=name&topic=DDE topic]/[ddeconnect]

Required Tag Parameters:

- program=application name (name of the DDE server program to connect to, such as PCAuthorize)
- topic=DDE topic name (name of the DDE topic on which to send messages, such as GetBatches. Defined by the DDE server program; refer to that program's documentation for more information.)

To embed the results of a DDE command into a page, insert a DDEConnect context into a template then place [DDESend] contexts inside of that. The DDESend steps contained inside the context are executed, and any returned value is displayed in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the DDE is executed.

DDEConnect does nothing by itself; you must insert one or more [DDESend] contexts inside it to perform any real work. DDEConnect establishes a

connection to the DDE server program, and provides an environment for the DDESend contexts to do their work and return text results.

For example:

```
[DDEConnect program=PCAuthorize&topic=GetBatches]
[DDESend]Tela00001031 1[/DDESend]
[/DDEConnect]
```

In this example, the DDE command “GetBatches from PCAuthorize” is executed, and the results (a list of all the batch information for batch #1) is displayed. Notice that the white space between “Tela00001031” and “1” is actually a tab character, which is part of the PCAuthorize specification.

Caution: DDE has complete access to all DDE-aware programs on your web server. You can write a DDE context that can do damage to the machine or retrieve secret information. However, remote visitors to your web site have no way of executing their own DDE contexts remotely; they can only execute DDE commands inside a template file you have saved on your web server’s hard disk.

[DDESEND] CONTEXT

Syntax: [ddesend]text[/ddesend]

Result: Sends text to a DDE server program on the local machine.

To embed the results of a DDE command into one of your pages, insert a [DDEConnect] context into a template, and place [DDESend] contexts inside of that. The DDESend steps contained inside the context are executed, and any returned value is displayed in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the DDE is executed.

DDESend must be inside a containing [DDEConnect] context, otherwise it does not know to which DDE server program to send its text.

For example:

```
[DDEConnect program=PCAuthorize&topic=GetBatches]
[DDESend]Tela00001031 1[/DDESend]
[DDESend]Tela00001031 2[/DDESend]
[/DDEConnect]
```

In this example, the DDE command “GetBatches from PCAuthorize” is executed, and the results (a listing of all the batch information for batch #1

and batch #2) is displayed. Notice that the white space between “Tela00001031” and “1” is actually a tab character, which is part of the PCAuthorize specification.

Caution: DDE has complete access to all DDE-aware programs on your web server. You can write a DDE context that can do damage to the machine or retrieve secret information. However, remote visitors to your web site have no way of executing their own DDE contexts remotely; they can only execute DDE commands inside a template file you have saved on your web server’s hard disk.

[DOS] CONTEXT

Syntax: [dos]batch file[/dos]

Result: Returns the results of the DOS-style batch file contained in the context.

To embed the results of a DOS Batch file into one of your pages, insert a DOS context into the template. The DOS program contained inside the context is executed, and any returned value is displayed in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the batch file is executed.

For example:

```
<pre>
[DOS]dir c: [/DOS]
</pre>
```

In this example, the DOS command “dir c:” is executed, and the results (a listing of all the files at the root of C: drive) is displayed. The <pre> tags are used to format the results more nicely in an HTML page.

Caution: DOS has complete access to your hard disk and all programs on your web server. You can write a DOS context that erases the contents of your hard disk. However, remote visitors to your web site have no way of executing their own DOS contexts remotely; they can only execute DOS commands inside a template file you have saved on your web server’s hard disk.

[SHELL] CONTEXT

Syntax: [Shell]UNIX Shell Commands[/Shell]

Executes the UNIX shell commands contained in the context and displays the results.

To embed the results of a shell command into one of your pages, put a Shell context into a template. The shell commands contained inside the context are executed, and any returned value is displayed in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the shell command is executed.

For example, normally you would put the following text into a .TPL file on your server and use a web browser to link to it:

```
<pre>
[Shell]ls -l[/Shell]
</pre>
```

In this example, the shell command "ls -l" is executed, and the results (a list of all the files in the current directory) is displayed. The <pre> tags are used to format the results in an HTML page. The user privileges are the same as the WebDNA program itself (which is typically logged on as user nobody).



Caution! Shell has complete access to your hard disk and all programs on your web server. You can write a Shell context to erase the contents of your hard disk. However, remote visitors to your web site have no way of executing their own Shell contexts remotely. They can only execute shell commands inside a template file that you have saved on your web server's hard disk.

[ELAPSEDTIME] TAG

Syntax: [ElapsedTime]

Placing [ElapsedTime] in your template displays the elapsed time (in 60ths of a second) since the beginning of processing this page. Put one at the top of a page, and another at the bottom to see how long the entire page takes to process (subtract the 2 numbers to get the answer).

FLUSHCACHE COMMAND

Syntax: FlushCache

Result: Clears all cached templates from memory.

To flush WebDNA's template cache, use a web browser to link to a URL containing the FlushCache command. Whenever WebDNA receives a FlushCache command, it immediately removes any "memorized" pages from memory. Newer versions of any HTML or template files that have been modified on disk are used the next time a browser links to those pages.

For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=FlushCache>

A confirmation message appears after flushing all the templates.

Other ways to send the same command include:

HTML Source	Description
<code></code>	Hyperlink to WebDNA CGI
<code></code>	Hyperlink to WebDNA plug-in Action
<code><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="FlushCache"> <input type="submit"> </form></code>	Form-based command to plug-in Action (notice the template is part of the action)

[FLUSHDATABASES] TAG

Syntax: [FlushDatabases]

Writes all databases to disk and closes them all.

To flush WebDNA's databases, use a web browser to link to a URL containing the FlushDatabases command. Whenever WebDNA receives a FlushDatabases command, it immediately saves any modified databases to disk. Newer versions of any database files that have been modified on disk will be used the next time a database is referenced.

For example, normally you would link to a URL or form containing the following information:

<http://yourserver.com/xx.tpl?command=FlushDatabases>

A confirmation message appears after flushing all the databases.

Here are some other ways to send the same command:

HTML Source	Description
<code></code>	Hyperlink to WebDNA plug-in Action
<code><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="FlushDatabases"> <input type="submit"> </form></code>	Form-based command to plug-in Action (notice the template is part of the action).

FLUSHDATABASES COMMAND

Syntax: FlushDatabases

Result: Writes all databases to disk and closes them all.

To flush WebDNA's databases, use a web browser to link to a URL containing the FlushDatabases command. Whenever WebDNA receives a FlushDatabases command, it immediately saves any modified databases to disk. Newer versions of any database files that have been modified on disk will be used the next time a database is referenced.

For example (normally you would link to a URL or form containing the following information):

<http://yourserver.com/xx.tpl?command=FlushDatabases>

A confirmation message appears after flushing all the databases.

Other ways to send the same command include:

HTML Source	Description
<code></code>	Hyperlink to WebDNA plug-in

	Action
<pre><form method="POST" action="xx.tpl"> <input type="hidden" name="command" value="FlushDatabases"> <input type="submit"> </form></pre>	<p>Form-based command to plug-in Action (notice the template is part of the action)</p>

[INTERPRET] CONTEXT

Syntax: [interpret]Any Text[/ interpret]

Result: Interprets the enclosed [xxx] values using the WebDNA interpreter.

To interpret [xxx] values stored in a database field, enclose them in an [Interpret] context. For example, if you insert the text [date] into a field in a database (text1, for instance), then when you display [text1] in a template it only shows the literal text [date] - it does not try to interpret the contents of the field. WebDNA does one level of interpretation; use the [Interpret] context to force it to re-interpret the results multiple times.

A good example is a banner advertisement database: Assume you want to create a list of advertisements to be displayed randomly at the top of every page in your site. This functions correctly if the ads themselves don't contain any [xxx] tags. However, if you want to insert cart information into the banner ad, you must use the [Interpret] context to tell the [Cart] variable to substitute its real value.

For example:

```
BannerAds.db
AdNumber BannerHTML
1 <a href="Invoice.tpl?cart=[cart]">Invoice</a>
2 
```

Review a sample template that incorrectly uses the database above to randomly choose 1 item from the list and display the results at the top of a page:

```
<html>
<body>
[search db=BannerAds.db&neAdNumberdata=0&
max=1&AdNumbersdir=ra&AdNumbersort=1]
```

```
[foundItems][BannerHTML][foundItems]
[/search]
</body>
</html>
```

The problem here is that [BannerHTML] is indeed evaluated and displayed, but it is incorrect because the [Random] tag inside the field hasn't been evaluated to its true value: .

The following sample template correctly uses the database above to randomly choose one item from the list and display the results at the top of a page:

```
<html>
<body>
[search db=BannerAds.db&neAdNumberdata=0
&max=1&AdNumbersdir=ra&AdNumbersort=1]
[foundItems][interpret][BannerHTML][interpret][foundItems]
[/search]
</body>
</html>
```

The results here are accurate: .

[OBJECT] CONTEXT

Syntax: [Object Parameters]Main Parameter[/Object]

Result: Executes the external Windows ActiveX control or Java class file, and displays the text of the result.

To embed the results of an external function (which could be a Windows ActiveX DLL or Java class file on any platform) into one of your pages, put an Object context with appropriate parameters into a template. The parameters are sent to the external module, and the results of the external call are displayed as text in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the Object is executed.

For example, normally you would put the following text into a .TPL file on your server and use a web browser to link to it:

```
[OBJECT objname=MS.CUSTOM[!]  
[!]&call=MyFunction&param1=Hello&param1type=text&param2=2000&param2ty  
pe=num]  
[/OBJECT]
```

The ActiveX control DLL “MS.CUSTOM” will be loaded and “MyFunction” will be executed with the following parameters:

- Parameter 1: Hello Type: text
- Parameter 2: 2000 Type: num

Parameter	Description
objname	ActiveX: The name of the ActiveX control Java: The name of the Java class file
call	ActiveX: Name of function to call. Java: Name of the method to call (Note: the method must be able to receive “java/lang/String” and return “java/lang/String”)
type	(Optional) The type of module to execute. 0 - ActiveX (Default) 1 - Java .class file
classpath	(Required for Java) Location of the .class file and all the supporting .jar files. Can contain multiple locations separated by semicolons “;”. Only java modules require this parameter. Note: This is only used on Mac OS platforms and is ignored in all the other platforms. If you want to set the classpath for the other platforms, you have to manually change the JavaClassPath preference in the "WebCatalog Prefs" file, but make sure you shutdown WebDNA before doing this.
Param1	(Optional) Value of the first parameter to be passed into the ActiveX or Java function. For boolean values, use 0 for FALSE and 1 for TRUE
Param1Type	(Required for each parameter) Data type of the first parameter. Choose from bool, num, or text
Param2	(Optional) Value of the second parameter to be passed into the ActiveX or Java function
Param2Type	(Required for each parameter) Data type of the

Parameter	Description
	second parameter
...ParamN	(Optional) As many parameters as are necessary may be passed into the ActiveX/Java function
...ParamNType	(Required for each parameter) For each value passed, you must define its data type

RAW COMMAND

Syntax: Raw

Result: Returns the “raw” contents of a template.

Sometimes you, or someone from technical support, needs to view the text of a template file in its raw form without being processed by WebDNA. Usually this is so you can see the WebDNA commands rather than the final resulting HTML those WebDNA commands generate. WebDNA normally intercepts URLs leading to template files and makes it impossible to “view the source” of a WebDNA template.

The Raw command is provided for sites using Suffix Mapping to automatically process all files ending in a particular extension. In this scenario, it is impossible to link to the file directly to view the raw contents of the file. While this is good because it does not allow the outside world to see the contents of your pages, it can make site development difficult if you are debugging a particular page remotely. The Raw command should be protected with the Admin password so it is not available to everyone accessing your site.

[REDIRECT] TAG

Syntax: [Redirect url=URL]

Placing [Redirect <http://www.webdna.us/>] in your template forces the remote browser to immediately ‘jump’ to the new location specified, rather than displaying whatever is in the template. Any other text in the template will be ignored.

[RETURNRAW] CONTEXT

Syntax: [returnraw]HTTP/1.0 200 OK...[/returnraw]

Result: Returns the raw HTML within the tags, including all MIME headers.

The [ReturnRaw] context allows you to return HTML with custom MIME headers. Normally, WebDNA returns a page using standard MIME headers. However, there are situations where you may want to return something different. An example is when you want to automatically re-direct a visitor to another page without requiring them to click a hypertext link.

For example:

```
[returnraw]HTTP/1.0 302 Found
Location: http://www.webdna.us
[/returnraw]
```

When this WebDNA is encountered in a template file, WebDNA immediately stops processing and returns the MIME headers and HTML contained within the context. HTML found before or after the context is not returned. In this example, the visitor is automatically redirected by the browser to the alternate location specified. In this respect, it is similar to the [Authenticate] tag, which immediately stops execution and returns.

Note that MIME headers normally require a linefeed (or carriage return / linefeed combination) after each line. If you are creating a template on a Macintosh, which normally places a carriage return by itself as the end-of-line marker, you will need to past the linefeed manually, or change the document type to DOS or UNIX.

The [ReturnRaw] context can be used to create custom authentication schemes as well.

[SPAWN] CONTEXT

Syntax: [spawn]WebDNA text[/spawn]

Result: Creates a new thread to execute WebDNA simultaneously with the current template.

To perform WebDNA simultaneously, place it inside a Spawn context. All WebDNA inside the Spawn context begins to execute immediately, and the remainder of the template is returned to the visiting browser immediately.

Note: The HTML output from within a Spawn context is never displayed to the browser. While this may seem unhelpful at first, realize that the purpose of Spawn is to allow you to execute very lengthy operations without forcing the visitor to wait for them. The WebDNA in the spawned context could update a database several minutes later, wait for a 15-second credit card operation, create a WebDelivery file, or many other useful things.

For example:

```
Before the spawn [elapsedtime]<br>
[Spawn]
-- Some WebDNA that takes a long time to finish
[Loop start=1&end=5000][ShowIf 1=1][ShowIf][Loop]
[/Spawn]
After the spawn [elapsedtime]<br>
```

The example above yields:

```
Before the spawn 1
After the spawn 3
```

Notice that the elapsedtime is very small even though the loop inside the spawn could take several seconds. This is because your web browser sees the results of the template before the spawned WebDNA is finished.

[TCPCONNECT] CONTEXT

Syntax: [tcpconnect host=domain
name&port=number][tcpSend][tcpconnect]

Result: Connects to a TCP port of another computer on the internet.

Required Tag Parameter: host=domain name (Name or IP address of the machine to connect to. Do not insert http or ftp into the host text. TCPConnect is a very low-level connection, and it does not understand these protocols.)

Optional Tag Parameter: port=number (TCP port number to connect to. If not specified, then 80 is assumed.)

To embed the results of a TCP session into one of your pages, insert a TCPConnect context into a template, and place [TCPSend] contexts inside of that. The TCPSend steps contained inside the context are executed, and any returned value is displayed in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the TCPSend is executed.

TCPConnect does nothing by itself; you must insert one or more [TCPSend] contexts inside it to perform any real work. TCPConnect establishes a connection to the TCP server program, and provides an environment for the TCPSend contexts to do their work and return text results.

For example:

```
[TCPConnect host=www.webdna.us&port=80]
[TCPSend]GET / HTTP/1.0[UnURL]%0D%0A%0D%0A[/TCPSend]
[/TCPConnect]
```

In this example, the http command equivalent to the URL "<http://www.webdna.us/>" is executed, and the results (the home page for that site) are displayed. Notice the use of [UnURL] to send <Carriage Return><LineFeed> <Carriage Return><LineFeed> as part of the TCPSend text. If you do not send the correct sequence of 2 CR/LF characters, the remote web server will never return any text, and the TCPSend will timeout while waiting for a response.

[TCPSEND] CONTEXT

Syntax: [tcp send]text[/tcp send]

Result: Sends text to a TCP server program on a remote machine.

Optional Tag Parameters:

- end=text (Text to look for that indicates end of line.)
- skipheader=T - instructs the WebDNA engine to 'strip' the MIME headers from the result

Often [TCPSend end=%0D%0A] is used to look for <carriage return><line feed> as the end of line indicator, as it is necessary when communicating with email POP servers or FTP servers. This is not necessary for communication with the http protocol because the remote server disconnects automatically at the end of the session.

To embed the results of a TCP session into one of your pages, insert a [TCPConnect] context into the template, and place [TCPSend] contexts inside of that. The TCPSend steps contained inside the context are executed, and any returned value is displayed in place of the context. Any [xxx] tags inside the context are first substituted for their real values before the TCPSend is executed.

TCPConnect does nothing by itself; you must insert one or more [TCPSend] contexts inside it to perform any real work. TCPConnect establishes a connection to the TCP server program, and provides an environment for the TCPSend contexts to do their work and return text results.

For example:

```
[TCPConnect host=www.webdna.us&port=80]
[TCPSend]GET / HTTP/1.0[UnURL]%0D%0A%0D%0A[/TCPSend]
[/TCPConnect]
```

In this example, the http command equivalent to the URL "<http://www.webdna.us/>" is executed, and the results (the home page for that site) are displayed. Notice the use of [UnURL] to send <Carriage Return><LineFeed> <Carriage Return><LineFeed> as part of the TCPSend text. If you do not send the correct sequence of 2 CR/LF characters, the remote web server will never return any text, and the TCPSend will timeout while waiting for a response.

Using 'SKIPHEADER=T' in the [TCPSend] context will instruct the WebDNA engine to 'strip' the MIME headers from the result (assuming that you are invoking an HTTP Get or Post).

For example:

A simple HTTP GET without the SKIPHEADER set to 'T'

```
[text]host=[listmimeheaders
name=host&exact=false][value][/listmimeheaders][/text]

[tcpconnect host=[host]][tcpsend]
GET [thisurl]?get=false HTTP/1.0
HOST: [host]

[/tcpsend][/tcpconnect]
```

Result:

```
HTTP/1.0 200 OK
Content-type: text/html
Content-Length: 29
```

Hello World!

With the 'SKIPHEADER=T' parameter...

```
[tcpconnect host=[host]][tcp send skipheader=T]
GET [thisurl]?get=false HTTP/1.0
HOST: [host]

[/tcp send][ /tcp connect]
```

Results:

Hello World!

[VERSION] TAG

Syntax: [Version]

Placing [Version] in your template displays the version of the currently running WebDNA program or plug-in.

Browser Info

[BROWSERNAME] TAG

Syntax: [BrowserName]

Specifying [BrowserName] in your document displays the name of the remote browser program running on the visitor's computer.

Placing [BrowserName] in your template displays the name of the remote browser program running on the visitor's computer.

[GETCOOKIE] TAG

Syntax: [GetCookie name=cookieName]

Placing [GetCookie fred] in your template displays the value of the cookie named "fred" that the remote browser has remembered. If no cookie of that

name exists, then nothing is returned. Use [ListCookies] to see all the cookies your browser is sending.

[GETMIMEHEADER] TAG

Syntax: [GetMimeHeader name=*headerName*]

Placing [GetMimeHeader Accept-Language] in your template displays the value of the MIME header called “Accept-Language,” which is a code representing the human language that the viewer is able to read. Use [ListMIMEHeaders] to see all the headers your browser is sending.

[IPADDRESS] TAG

Syntax: [IPAddress]

Placing [IPAddress] in your template will display the ip address of the remote computer that the visitor is using to view your site. All sections of the IP address are expanded to 3 digits, so that 207.67.2.14 will display as 207.067.002.014. This helps make partial comparisons inside [ShowIf] tags work more easily.

[ISSECURECLIENT] TAG

Syntax: [IsSecureClient]

Note: This tag will be supported through version 4.0. It will be eliminated in future versions.

Placing [IsSecureClient] in your template replaces the [IsSecureClient] tag with “T” if the remote browser is capable of SSL (Secure Socket Layer) connections, and “F” if not. As new browsers are released, you can control which ones are reported as secure by changing the “Browser Info.txt” file to indicate what level of HTML it supports, and whether or not it supports SSL.

[LISTCOOKIES] CONTEXT

Syntax: [listcookies][name][/listcookies]

Result: Lists all the cookies that are available to the current page from the visitor’s browser.

Optional Tag Parameters:

- **name** – The name of the cookie to retrieve. This is either an exact name or partial name depending upon the value of exact described below.
- **Exact** – Whether the value used in name should match the cookie name exactly or partially. Possible values are “T” (True) or “F” (False). The default value is “T”.

Optional Context Parameters:

- **name** – The name of the cookie being listed
- **value** – The value of the cookie being listed.
- **index** – The index (starting with 1) of the cookies being listed.

Cookies are values that can be set in a visitors browser by the server in order to save information that will identify or describe the visitor while they are using your site. Cookies require the consent of the visitor to be set and should not be relied upon unless so stated to your visitors. Cookies provide an alternate way to store the cart value, for example that doesn't require you to pass that value through all your URL's.

Consult an HTTP reference for more information on cookies.

For example:

```
The following are all the cookies available to this page:<br>
[ListCookies]
[index],[name],[value]<br>
[/ListCookies]
```

Setting Cookies

You set cookies by returning additional MIME header information to the browser. This can be done currently using the [returnraw] context. The following example sets a cookie whose name is “text” and whose value is “abcd”.

For example:

```
[returnraw]HTTP/1.0 200 OK
Set-Cookie: text=abcd; expires=Wednesday, 01-Dec-1999
23:23:23 GMT; path=/; domain=www.yourdomain.com
Cookie set.
[/returnraw]
```

Note that you must set an expiration date and domain/path from which you can retrieve the cookie as well. This example returns the text “Cookie Set”. However, it is more likely that you’ll want to return an entire HTML template. To do this, redirect the visitor at the same time you set the cookie.

```
[returnraw]HTTP/1.0 302 Found
Location: http://www.yourdomain.com/entry.html
Set-Cookie: text=abcd; expires=Wednesday, 01-Dec-1999
23:23:23 GMT; path=/; domain=www.yourdomain.com
[/returnraw]
```

MIME headers require a linefeed as a line separator (or carriage return / linefeed combination). If you are using a text editor on the Macintosh you must paste a linefeed in after each line or set the file to a DOS or UNIX mode.

[LISTMIMEHEADERS] CONTEXT

Syntax: [listmimeheaders][name],[value][/listmimeheaders]

Result: Lists all the MIME headers that were used to get the current page.

Optional Tag Parameters:

- **name** – The name of the MIME header to list. This can be an exact or partial name depending on the value of the exact parameter.
- **exact** – Possible values are “T” (True) or “F” (False). This parameter determines whether the value of the name parameter must match exactly or find a partial match. The default value (if this parameter is not listed) is “T”.

Optional Context Tags:

- **[name]** – The name of the MIME header.
- **[value]** – The value of the MME header.
- **[index]** – The index number of the current MIME header (starting with 1).

MIME headers contain technical information describing the HTML, and other miscellaneous information being returned. Many different MIME headers exist and it is beyond the scope of this document to describe them all. Refer to an HTTP technical reference for more information.

Example 1:

The following are all the MIME headers available to this page:


```
[listmimeheaders]
[index],[name],[value]<br>
[/listmimeheaders]
```

Example 2:

```
Your web browser preferences show that your native language is:<br>
[listmimeheaders name=HTTP_ACCEPT_LANGUAGE]
[value]
[/listmimeheaders]
```

Example 3:

```
The following are the values of all the MIME headers whose name begins with
"HTTP_ACCEPT":<br>
[ListMIMEHeaders name=HTTP_ACCEPT&exact=F]
[index],[value]<br>
[/listmimeheaders]
```

[REFERRER] TAG

Syntax: [Referrer] or [Referer]

Placing [Referrer] in your template displays the URL of the referring page that led to this one. This is a handy way to give visitors a "back up" button.

Note: This will not work if the previous page was a FORM METHOD="POST".

[SETCOOKIE] TAG

Syntax: [SetCookie name=cookieName&value=cookieValue&expires=expireDate&path=/&domain=www.yourdomain.com]

Placing a [SetCookie] tag into a template causes the remote browser to create or replace a cookie of that name in its local list of cookies. You can use [GetCookie] or [ListCookies] later to retrieve that value from the remote browser. The expiration date of the cookie must be of the form Wednesday, 09-Nov-1999 23:12:40 GMT. The domain must be the name of your web server, otherwise the browser will not provide the cookie information.

Example:

```
[SETCOOKIE name=Visits&value=12&expires=Wednesday, 09-Nov-1999
23:12:40 GMT&path=/&domain=www.webdns.us]
```

Or one that expires 7 days from today (the +05:00:00 handles the US East Coast time difference from GMT):

```
[SETCOOKIE name=cookieName&value=sample&expires =[format
days_to_date %A, %d-%b-%Y][math]{{date}}+7[/math][format]
[math time]{{time}}+{05:00:00}[/math]
GMT&path=/&domain=www.yourserver.com]
```

Note: If you do not specify an **expires** parameter, then the cookie becomes a “session cookie,” expiring automatically when the browser quits.

[SETMIMEHEADER] TAG

Syntax: [SetMIMEHeader **name**=headerName&**value**=headerValue]

Placing a [SetMIMEHeader] tag into a template causes WebDNA to add a new MIME header to the outgoing HTML text for that page. MIME headers are normally used to create redirect requests and cookies. WebDNA already has special tags for generating redirects and cookies, but in the future you may need to create MIME headers for other purposes.

XML

[XMLPARSE] CONTEXT

New in 5.0

Syntax: [xmlparse var=...&source=...]<optional inline XML content>[/xmlparse]

Result: Enable the WebDNA programmer to input XML data into a WebDNA variable, after which any part of the XML structure can be readily examined using the new WebDNA XML contexts.

Optional Tag Parameters:

- **var** - The variable name you specify to represent the XML parsed object.
- **source** - A URL reference to an external XML file. If 'source' is provided, the content between the [xmlparse...] and [/xmlparse] tags is ignored.

Now lets parse this file into a WebDNA variable called 'xml_var1'...

We use the following code...

```
[xmlparse var=xml_var1][include  
  file=example1.xml][/xmlparse]
```

Note that the **[include]** tag was used to place the xml file contents between the xmlparse tags. We could just as easily 'pasted' the xml contents there as well.

Internally, the WebDNA engine 'saves' this parsed instance as an internal variable named 'xml_var1'. This saved instance can then be referenced in the other WebDNA XML contexts.

Results...

If the parse succeeded, you should not see any error messages between the two lines shown above.

The default behavior of this context is to iterate the child XML nodes of a parent node. The location of the parent node, in the xml 'tree', is determined by the 'path' parameter. If a path parameter is not provided, then the child nodes of the 'ref's root are iterated.

The path parameter can take three different forms: **'named:', 'indexed:', or 'xpath:'**.

- The **'named:'** method expresses a literal path to the parent node, i.e. **'path=named:CATALOG/CD(n)'**. If there are more than one similarly named 'sibling' nodes, then the '(n)' part specifies which node to select as part of the path.

- The **'indexed:'** method expresses an numerical 'step' wise path to the parent node, i.e. **'path=indexed:1/2/3'**. This example could be expressed as: 'The third child node of the second child node of the first child node of the xml root'.

- The **'xpath:'** method is an XPath 'expression' that evaluates to a collection of nodes in the XML tree. In this case, the iterated nodes are those of the

resulting 'collection' of nodes. This is a bit different from the 'named' and 'indexed' method in that the collection of node are not the 'child' nodes of a given 'parent' node. This is the most powerful method for selecting XML nodes. There are several online 'xpath' tutorials that you can visit that will help you develop your XPath skills.

We will be using the 'named' method in the next few tutorial pages.

Now lets use the **[xmlnodes]** context to iterate the xml child nodes of the root xml node of the 'example1.xml' document.

The code is as follows...

```
[xmlparse var=xml_var1][include
  file=example1.xml][/xmlparse]

[xmlnodes ref=xml_var1]
[name]=[value]

[/xmlnodes]
```

Results...

```
CATALOG=
```

We see that the 'CATALOG' node is the only child node from the root of the xml file. Notice that the 'value' is empty. This is because the 'CATALOG' node has no value, and is actually a 'container' node for other xml nodes. So a 'value' will only be displayed for a 'leaf' xml node, i.e.

JOHN

In this case, [name] would evaluate to 'FIRSTNAME' and [value] would be 'JOHN'.

Lets dive a little deeper into the xml file and iterate the 'child' nodes of the root 'catalog' node...

We now use...

```
[xmlparse var=xml_var1][include
  file=example1.xml][xmlparse]

[xmlnodes ref=xml_var1&path=named:Catalog]
[name]=[value]

[/xmlnodes]
```

Results....

```
CD=  
  CD=  
    CD=  
      CD=
```

We see that we have iterated all the 'CD' child nodes of the 'Catalog' parent node. Again, none of the resulting child nodes contain a value as they are all 'container' nodes.

You can embed any number of [xmlnodes] contexts within each other. Lets do this to iterate the child nodes of all the 'CD' nodes...

We use...

```
[xmlparse var=xml_var1][include  
  file=example1.xml][/xmlparse]  
  
[xmlnodes ref=xml_var1&path=named:Catalog]  
[name] - [index]  
  
[xmlnodes]  
- [name]=[value]  
  
[/xmlnodes]  
[/xmlnodes]
```

Results....

```
CD - 1  
- TITLE=Empire Burlesque  
- ARTIST=Bob Dylan  
- COUNTRY=USA  
- COMPANY=Columbia  
- PRICE=10.90  
- YEAR=1985  
CD - 2  
- TITLE=Hide your heart  
- ARTIST=Bonnie Tylor  
- COUNTRY=UK  
- COMPANY=CBS Records  
- PRICE=9.90  
- YEAR=1988
```

```

CD - 3
- TITLE=Greatest Hits
- ARTIST=Dolly Parton
- COUNTRY=USA
- COMPANY=RCA
- PRICE=9.90
- YEAR=1982
CD - 4
- TITLE=Still got the blues
- ARTIST=Gary More
- COUNTRY=UK
- COMPANY=Virgin records
- PRICE=10.20
- YEAR=1990

```

Now we are getting some interesting results. Note that the 'inner' xmlnodes context did not need a 'ref' parameter. This is because the inner xmlnodes context inherited the outer xmlnodes' current iterated node. Also notice that the inner xmlnodes context did not use a 'path' parameter. So it uses the 'root' path of the outer xmlnodes' current iterated node.

Did you also notice the use of the [index] tag in the outer xmlnodes context? As with most 'iterative' WebDNA contexts, the [index] tag resolves to the current iteration 'count'.

Lets refine the named path parameter to go directly to a particular 'CD' node.

With a minor change to the 'path' parameter, we can retrieve all the child nodes of the fifth 'CD' node.....

```

[xmlparse var=xml_var1][include
  file=example1.xml][/xmlparse]

[xmlnodes ref=xml_var1&path=named:Catalog/CD(5)]
[name]=[value]

[/xmlnodes]

```

Results....

```

TITLE=Eros
ARTIST=Eros Ramazzotti
COUNTRY=EU
COMPANY=BMG

```

```
PRICE=9.90
YEAR=1997
```

Using the 'name' parameter we can filter the results to display only the 'TITLE' node of the fifth 'CD' node.....

```
[xmlparse var=xml_var1][include
  file=example1.xml][/xmlparse]

[xmlnodes ref=xml_var1&path=named:Catalog/CD(5) &name=TITLE]
[name]=[value]

[/xmlnodes]
```

Results....

```
TITLE=Eros
```

Using the 'name' and 'exact' parameters, we can filter the results to display only those nodes, of the fifth 'CD' node, where the node name matches a given sub-string, 'CO'.....

```
[xmlparse var=xml_var1][include
  file=example1.xml][/xmlparse]

[xmlnodes
  ref=xml_var1&path=named:Catalog/CD(5) &name=CO&exact=F]
[name]=[value]

[/xmlnodes]
```

Results....

```
COUNTRY=EU
COMPANY=BMG
```

[XMLNODES] CONTEXT

New in 5.0

Syntax: [xmlnodes ref=...&path=...&name=...&exact=T/F]
<WebDNA>[/xmlnodes]

Result: Lists all the form variables and parameters passed to the current page.

Optional Tag Parameters:

- **ref** - Reference to an xml parsed object variable. If this parameter is not provided, then it is assumed that there is an 'outer' [xmlnodes] context from which to reference a particular XML node. (This is explained further on in this tutorial).
- **path** - Depending on which path method is used (see below), this will either be a path to the parent xml element (node) from which to iterate the child elements (nodes), or a path 'expression' representing a collection of XML nodes. If a path is not provided, then the child nodes of the 'ref' node will be iterated
- **name** - A string used to filter the resulting xml nodes. Only the xml nodes that match the 'name' string will be iterated.
- **exact** - Used with the 'name' parameter. Either 'T' or 'F'. Specifies whether the 'name' parameter is a 'whole' string match or a 'sub-string' match.

Optional Context Tags:

- **[name]** - The name of the current iterated XML node.
- **[value]** - The value of the current iterated XML node. Will be empty if the node is a 'container' node, i.e. contains other XML nodes.
- **[index]** - The 'count' of the current iterated node.
- **[numfound]** - The total 'count' of iterated nodes.
- **[content]** - The 'raw' xml content of the current node.
- **[iscontainer]** - 'T' if the current node contains other XML nodes.

This context is used to retrieve the contents of the specific node. The 'path' parameter is used to locate the node. As with the [XMLNodes] context, the 'path' parameter has three modes; **'named:', 'indexed:', and 'xpath:'**.

[XMLNode] can also be used to persist a 'pointer' to a specific node. This reference can then be used in subsequent calls to other XML contexts.

Lets use the XMLNode context to retrieve the TITLE information of the third CD node....

Code used...

```
[xmlparse var=xml_var1][include  
file=example1.xml][/xmlparse]
```

```
[xmlnode ref=xml_var1&path=indexed:1/3/1]
[name]=[value]

[/xmlnode]
```

You'll notice that we used the 'indexed' path method. This is because we have explicit knowledge of the XML file, and can therefore use the indexed method to jump quickly to the desired XML node.

Results...

```
TITLE=Greatest Hits
```

Now let's use the **[XMLNode]** context to persist a reference to the third 'CD' node, then use that reference in an XMLNodes context to retrieve the child nodes...

Code used...

```
[xmlparse var=xml_var1][include
file=example1.xml][/xmlparse]

[xmlnode
ref=xml_var1&path=indexed:1/3&var=xml_CD3][xmlnode]

[xmlnodes ref=xml_CD3]
[name]=[value]

[/xmlnodes]
```

Results...

```
TITLE=Greatest Hits
ARTIST=Dolly Parton
COUNTRY=USA
COMPANY=RCA
PRICE=9.90
YEAR=1982
```

[XMLNODESATTRIBUTES] CONTEXT

New in 5.0

Syntax: [XMLNodeAttributes ref=...&path=...]<WebDNA>

[/XMLNodeAttributes]

Result: This context is used to iterate the attributes of a specific XML node. The 'path' parameter is used to locate the node. As with the [XMLNodes] context, the 'path' parameter has three modes; **'named:'**, **'indexed:'**, and **'xpath:'**.

Optional Tag Parameters:

- **ref** - Reference to an xml object variable. If this parameter is not provided, then it is assumed that there is an 'outer' [xmlnode/s] context from which to reference a particular XML node.
- **path** - Path to the desired XML node. If an XPath expression is used, it should evaluate to a single node.

Optional Context Tags:

- **[name]** - The name of the current iterated XML node attribute.
- **[value]** - The value of the current iterated XML node attribute.
- **[index]** - The 'count' of the current iterated attribute.
- **[numfound]** - The total 'count' of iterated attributes.

Lets use the **[XMLNodeAttributes]** context to retrieve the attributes of the first CD node...

```
[xmlparse var=xml_var1][include
  file=example1.xml][/xmlparse]
```

Attributes for the 'CD' node:

```
[xmlnode ref=xml_var1&path=indexed:1/1]
[content]
[/xmlnode]
Are...
```

```
[xmlnodeattributes ref=xml_var1&path=indexed:1/1]
[name]=[value]

[/xmlnodeattributes]
```

Results...

Attributes for the 'CD' node:

```
Bob Dylan
USA
Columbia
10.90
1985
```

```
Are...
id=123
status=instock
```

To simplify the code, we could place the **[XmlNodeAttributes]** context 'inside' of the **[XMLNode]** context. In this case, we will not need to supply the 'path' or 'ref' parameters to **[XmlNodeAttributes]**, since it will use the 'implied' XML node in the outer **[XMLNode]** context.

We use...

```
[xmlparse var=xml_var1][include
  file=example1.xml][/xmlparse]

Attributes for the 'CD' node:
[xmlnode ref=xml_var1&path=indexed:1/1]
[content]
Are...

[xmlnodeattributes]
[name]=[value]

[/xmlnodeattributes]
[/xmlnode]
```

Results...

```
Attributes for the 'CD' node:

Bob Dylan
USA
```

```
Columbia
10.90
1985

Are...
id=123
status=instock
```

[XSL] CONTEXT

New in 5.0

Syntax: [xsl var=...&source=...][/xmlparse]

Result: Enables the WebDNA programmer to compile and apply XSL style sheets to XML data, allowing the programmer to query and render XML data as they see fit.

Optional Tag Parameters:

var - The variable name you specify to represent the XSL parsed object.

source - A URL reference to an external XSL file. If 'source' is provided, the content between the [xsl...] and [/xsl] tags is ignored.

This will explain how to use the new **WebDNA XSL/XSLT contexts**. This assumes the user is familiar with the WebDNA language.

It is recommended that you view 'XML Contexts' before continuing with this.

Also...

If you plan on using the [XSL] or [XSLT] WebDNA contexts, you will need to become familiar with the **XPath** and **XSLT** languages:

XSLT is a language for transforming XML documents into other XML documents.

XPath is a language for defining parts of an XML document.

There are several online tutorials that will help. Just use your favorite search engine and search for 'XSL XSLT XPATH Tutorials'.

These new contexts enable the WebDNA programmer to compile and apply XSL style sheets to XML data, allowing the programmer to query and render XML data as they see fit.

[XSL]

You can pre-compile XML style sheet code into a WebDNA variable, and then apply the XSL object to XML data via the XSLT context.

Compile XSL

Lets compile some XSL source into a WebDNA variable called 'xsl_var1'...

We use the following code...

```
[xsl var=xsl_var1]
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
[/xsl]
```

Internally, the WebDNA engine 'saves' this compiled instance as an internal variable named 'xsl_var1'. This saved instance can then be referenced when using the XSLT context.

Results...

If the operation succeeded, you should not see any error messages between the two lines shown above.

[XSLT] CONTEXT

New in 5.0

Syntax: [XSLT xslref=...(or xslsource=...)&xmlsource=][XSLT]

Result: The [XSLT] Context allows the WebDNA programmer to 'apply' an XSL style sheet to an XML document and thus 'transform' the XML data into any format the programmer desires (usually HTML).

Optional Tag Parameters:

xslref - named reference to a previously compiled block of XSL source.

xslsource - URL path to an external XSL document. This is an alternative to using the xslref parameter. Meaning that it is possible to use the [XSLT] context without having used the [XSL] context to pre-compile XSL source code. However, if you plan to use the same XSL code several times throughout the template, then it would be more efficient to compile the XSL source code once, using [XSL], then just use the XSL object reference when needed.

xmlsource - URL reference to an external XML document on which to apply the XSL transformation. If this parameter is supplied, then any in-line XML code that exists between the [XSLT] tags will be ignored.

Now that you know how to compile and persist XSL code, lets take a look at the XSLT context.

Lets use the previous XSL example to compile XSL code and then apply it to an XML document...

Example - Select all the 'CD' nodes in the example1.xml XML document and transform the results into an HTML table, with the 'text' data of each CD node tree displayed in the table rows.

Here is the code...

```
[xsl var=xsl_var1]
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
[/xsl]

[xslt xslref=xsl_var1][include file=example1.xml][/xslt]
```

Results...

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tylor
Greatest Hits	Dolly Parton
Still got the blues	Gary More
Eros	Eros Ramazzotti
One night only	Bee Gees

Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers
Big Willie style	Will Smith
Tupelo Honey	Van Morrison
Soulsville	Jorn Hoel
The very best of	Cat Stevens
Stop	Sam Brown
Bridge of Spies	T`Pau
Private Dancer	Tina Turner
Midt om natten	Kim Larsen
Pavarotti Gala Concert	Luciano Pavarotti
The dock of the bay	Otis Redding
Picture book	Simply Red
Red	The Communards
Unchain my heart	Joe Cocker

This is a pretty simple table, but it should give you an idea of what you can do with XSL(T).

Here is the same example, but without using the XSL context to pre-compile the in-line XSL code. Instead, a URL reference is made to an external xsl file, which happens to match the XSL code block above. It also uses the 'xmlsource' parameter to reference an external xml file to transform.

Here is the source...

```
[!] build the URL path to this lab folder [!]
[text]host=[listmimeheaders
  name=HOST&exact=F][value][!listmimeheaders][!text]
[text]path=[url][listpath
  pathonly=T&path=[thisurl]][name][!listpath][!url][!text]
]
```

```
[xslt
  xslsource=http://[host]/[path]example1.xsl&xmlsource=htt
  p://[host]/[path]example1.xml[/xslt]
```

Results (should be the same as above)...

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tylor
Greatest Hits	Dolly Parton
Still got the blues	Gary More
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers
Big Willie style	Will Smith
Tupelo Honey	Van Morrison
Soulsville	Jorn Hoel
The very best of	Cat Stevens
Stop	Sam Brown
Bridge of Spies	T`Pau
Private Dancer	Tina Turner
Midt om natten	Kim Larsen
Pavarotti Gala Concert	Luciano Pavarotti
The dock of the bay	Otis Redding
Picture book	Simply Red
Red	The Communards
Unchain my heart	Joe Cocker

This xsl example will display all the CD 'Titles' and 'Artists' and add a pink background-color to the artist column WHEN the price of the cd is higher than 10.

Here is the code...

```
[xsl var=xsl_var1]
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<xsl:choose>
<xsl:when test="PRICE>'10'">
<td bgcolor="#ff00ff">
<xsl:value-of select="ARTIST"/></td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="ARTIST"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
[/xsl]

[xslt xslref=xsl_var1][include file=example1.xml][xslt]
```

Results...

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan

Hide your heart	Bonnie Tylor
Greatest Hits	Dolly Parton
Still got the blues	Gary More
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers
Big Willie style	Will Smith
Tupelo Honey	Van Morrison
Soulsville	Jorn Hoel
The very best of	Cat Stevens
Stop	Sam Brown
Bridge of Spies	T`Pau
Private Dancer	Tina Turner
Midt om natten	Kim Larsen
Pavarotti Gala Concert	Luciano Pavarotti
The dock of the bay	Otis Redding
Picture book	Simply Red
Red	The Communards
Unchain my heart	Joe Cocker

This example will display all the CD 'Titles' and 'Artists' sorting them by artist.

Here is the code...

```
[xsl var=xsl_var1]
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
```

```

<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<xsl:sort select="ARTIST"/>
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
[/xsl]

```

```
[xslt xslref=xsl_var1][include file=example1.xml][xslt]
```

Results...

My CD Collection

Title	Artist
Romanza	Andrea Bocelli
One night only	Bee Gees
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tylor
The very best of	Cat Stevens
Greatest Hits	Dolly Parton
Sylvias Mother	Dr.Hook
Eros	Eros Ramazzotti
Still got the blues	Gary More
Unchain my heart	Joe Cocker
Soulsville	Jorn Hoel
For the good times	Kenny Rogers
Midt om natten	Kim Larsen
Pavarotti Gala Concert	Luciano Pavarotti
1999 Grammy Nominees	Many

The dock of the bay	Otis Redding
When a man loves a woman	Percy Sledge
Maggie May	Rod Stewart
Stop	Sam Brown
Black angel	Savage Rose
Picture book	Simply Red
Bridge of Spies	T`Pau
Red	The Communards
Private Dancer	Tina Turner
Tupelo Honey	Van Morrison
Big Willie style	Will Smith

Append XML data to a database

In this example we use XSLT to generate WebDNA [replace] code to replace/append XML data to a WebDNA database. It is a little tricky embedding the WebDNA into the XSL code, but it can be done.

Here is the code...

```
[xsl var=xsl_var1]

[!]
[!]%5Breplace
  db=music.db[url]&[/url]eqTITLEdata=[url]&[/url]append=T%
  5D[!]
[!]TITLE=[url]&[/url][!]
[!]ARTIST=%5B/replace%5D

[/xsl]

[text]result=[unurl][xslt xslref=xsl_var1][include
  file=example1.xml][[/xslt][[/unurl][[/text]
```

The XSLT operation produced the following WebDNA code...

```
[result]
```

We then use the interpret context to 'parse' WebDNA code. And use the search context to confirm that the entries were added.

```
[!]clear out the database entries[!]  
[delete db=music.db&neTITLEdata=[blank]]  
  
[interpret]  
[result]  
[/interpret]  
  
[search db=music.db&neTITLEdata=[blank]]  
  
Found [numfound] CD entries in the music database.  
  
[founditems]  
[TITLE] - [ARTIST]  
  
[/founditems]  
[/search]  
  
[closedatabase db=music.db]
```

Results...

The XSLT operation produced the following WebDNA code...

```
[replace db=music.db&eqTITLEdata=Empire  
  Burlesque&append=T]TITLE=Empire Burlesque&ARTIST=Bob  
  Dylan[/replace]  
[replace db=music.db&eqTITLEdata=Hide your  
  heart&append=T]TITLE=Hide your heart&ARTIST=Bonnie  
  Tylor[/replace]  
[replace db=music.db&eqTITLEdata=Greatest  
  Hits&append=T]TITLE=Greatest Hits&ARTIST=Dolly  
  Parton[/replace]  
[replace db=music.db&eqTITLEdata=Still got the  
  blues&append=T]TITLE=Still got the blues&ARTIST=Gary  
  More[/replace]
```

```

[replace
  db=music.db&eqTITLEdata=Eros&append=T]TITLE=Eros&ARTIST=
  Eros Ramazzotti[/replace]
[replace db=music.db&eqTITLEdata=One night
  only&append=T]TITLE=One night only&ARTIST=Bee
  Gees[/replace]
[replace db=music.db&eqTITLEdata=Sylvias
  Mother&append=T]TITLE=Sylvias
  Mother&ARTIST=Dr.Hook[/replace]
[replace db=music.db&eqTITLEdata=Maggie
  May&append=T]TITLE=Maggie May&ARTIST=Rod
  Stewart[/replace]
[replace
  db=music.db&eqTITLEdata=Romanza&append=T]TITLE=Romanza&A
  RTIST=Andrea Bocelli[/replace]
[replace db=music.db&eqTITLEdata=When a man loves a
  woman&append=T]TITLE=When a man loves a
  woman&ARTIST=Percy Sledge[/replace]
[replace db=music.db&eqTITLEdata=Black
  angel&append=T]TITLE=Black angel&ARTIST=Savage
  Rose[/replace]
[replace db=music.db&eqTITLEdata=1999 Grammy
  Nominees&append=T]TITLE=1999 Grammy
  Nominees&ARTIST=Many[/replace]
[replace db=music.db&eqTITLEdata=For the good
  times&append=T]TITLE=For the good times&ARTIST=Kenny
  Rogers[/replace]
[replace db=music.db&eqTITLEdata=Big Willie
  style&append=T]TITLE=Big Willie style&ARTIST=Will
  Smith[/replace]
[replace db=music.db&eqTITLEdata=Tupelo
  Honey&append=T]TITLE=Tupelo Honey&ARTIST=Van
  Morrison[/replace]
[replace
  db=music.db&eqTITLEdata=Soulsville&append=T]TITLE=Soulsv
  ille&ARTIST=Jorn Hoel[/replace]
[replace db=music.db&eqTITLEdata=The very best
  of&append=T]TITLE=The very best of&ARTIST=Cat
  Stevens[/replace]
[replace
  db=music.db&eqTITLEdata=Stop&append=T]TITLE=Stop&ARTIST=
  Sam Brown[/replace]
[replace db=music.db&eqTITLEdata=Bridge of
  Spies&append=T]TITLE=Bridge of
  Spies&ARTIST=T`Pau[/replace]
[replace db=music.db&eqTITLEdata=Private
  Dancer&append=T]TITLE=Private Dancer&ARTIST=Tina
  Turner[/replace]

```

```

[replace db=music.db&eqTITLEdata=Midt om
natten&append=T]TITLE=Midt om natten&ARTIST=Kim
Larsen[/replace]
[replace db=music.db&eqTITLEdata=Pavarotti Gala
Concert&append=T]TITLE=Pavarotti Gala
Concert&ARTIST=Luciano Pavarotti[/replace]
[replace db=music.db&eqTITLEdata=The dock of the
bay&append=T]TITLE=The dock of the bay&ARTIST=Otis
Redding[/replace]
[replace db=music.db&eqTITLEdata=Picture
book&append=T]TITLE=Picture book&ARTIST=Simply
Red[/replace]
[replace
db=music.db&eqTITLEdata=Red&append=T]TITLE=Red&ARTIST=Th
e Communards[/replace]
[replace db=music.db&eqTITLEdata=Unchain my
heart&append=T]TITLE=Unchain my heart&ARTIST=Joe
Cocker[/replace]

```

We then use the interpret context to 'parse' WebDNA code. And use the search context to confirm that the entries were added.

Found 26 CD entries in the music database.

```

Bridge of Spies - T`Pau
The dock of the bay - Otis Redding
Black angel - Savage Rose
The very best of - Cat Stevens
Unchain my heart - Joe Cocker
Romanza - Andrea Bocelli
Midt om natten - Kim Larsen
Eros - Eros Ramazzotti
Red - The Communards
For the good times - Kenny Rogers
Soulsville - Jorn Hoel
Big Willie style - Will Smith
Maggie May - Rod Stewart
Stop - Sam Brown
Still got the blues - Gary More
Private Dancer - Tina Turner
When a man loves a woman - Percy Sledge
Pavarotti Gala Concert - Luciano Pavarotti
Hide your heart - Bonnie Tylor
Picture book - Simply Red
1999 Grammy Nominees - Many
One night only - Bee Gees
Empire Burlesque - Bob Dylan

```

Greatest Hits - Dolly Parton
Sylvias Mother - Dr.Hook
Tupelo Honey - Van Morrison

Miscellaneous

[ARRAYSET] CONTEXT

New in 5.0

Syntax: [arrayset name=...&dim=...]<WebDNA that resolves to array element assignments>[/arrayset]

Result: The array context will allow the WebDNA programmer to create an array data object with up to five dimensions.

Parameters:

- **Name** - The variable name you specify to represent the array object.
- **Dim** - Comma delimited series of numbers representing the array dimension sizes, i.e. '3,3,3', which would represent a 3x3x3 dimensioned array.

To create a three-dimensional array, insert a [arrayset] context into a template.

For example:

```
[arrayset name=myarray&dim=2,2,2]
(1,1,1)=aaa&(1,1,2)=aab&(1,2,1)=aba&(1,2,2)=abb
[/arrayset]
```

This creates an array, called 'myarray', that will persist for the duration of the template. Note that the 'index=value' pairs within the [arrayset] tags are optional when creating a new array. Array indices can be assigned at any time.

When the '**dim**' parameter is supplied with the arrayset context, it implies that this will be a new array object. Subsequent calls to arrayset, using the same variable name (and without using the 'dim' parameter, imply new index assignments to an existing array object.

So after we have created the 'myarray' object using the line shown above, we can assign index value at a later time using:

```
[arrayset name=myarray](2,2,2)=xxx&(2,2,2)=zzz[/arrayset]
```

[ARRAYGET] CONTEXT

New in 5.0

Once the array has been created, and index assignments have been made, the array index values can be retrieved in the following ways:

Using the [ArrayGet] context to retrieve several index values...

```
[arrayget name=myarray]
[loop start=1&end=2]
  [loop start=1&end=2]
    [loop start=1&end=2]
      Index [::::index],[::index],[index] =
        (::::index],[::index],[index])

    [/loop]
  [/loop]
[/loop]
[/arrayget]
```

[ArrayGet] takes just one parameter; 'name', in which you place an existing array variable name. The (a,b,c,d,e) patterns evaluate to the corresponding index values. These patterns can be intermingled with other text, as shown.

```
Index 1,1,1 = 1-1-1
Index 1,1,2 = 1-1-2
Index 1,2,1 = 1-2-1
Index 1,2,2 = 1-2-2
Index 2,1,1 = 2-1-1
Index 2,1,2 = 2-1-2
Index 2,2,1 = 2-2-1
Index 2,2,2 = 2-2-2
```

[NumDims] Tag

You can use the [NumDims] tag to retrieve the number of dimensions for an array. For example:

```
[ArraySet name=array_1&dim=3,3,10][/ArraySet]

array_1 contains [ArrayGet name=array_1][numdims][/ArrayGet]
dimensions.<br>
```

Results:

```
array_1 contains 3 dimensions.
```

You can also use the 'array name' global tag to retrieve the number of dimension in a named array object. Here is the code:

```
[ArraySet name=array_1&dim=3,3,10][/ArraySet]

array_1 contains [array_1(numdims)] dimensions.
```

Results:

```
array_1 contains 3 dimensions.
```

[DimSize_] Tag

The [DimSize_] tag is used to retrieve the size of a given array dimension.

Example:

```
[ArraySet name=array_1&dim=3,4,5][/ArraySet]

[ArrayGet name=array_1]
array_1 has [NumDims] dimensions:

[loop start=1&end=[NumDims]]
Dimension [index] has
  [interpret][DimSize_[index]][/interpret] indexes.

[/loop]
[/ArrayGet]
```

Results:

```
array_1 has 3 dimensions:
Dimension 1 has 3 indexes.
Dimension 2 has 4 indexes.
Dimension 3 has 5 indexes.
```

And, again, you can access the dimension sizes using the global 'array name' tag, as follows:

```
[ArraySet name=array_1&dim=3,4,5][/ArraySet]

array_1 has [array_1(NumDims)] dimensions:

[loop start=1&end=[array_1(NumDims)]]
Dimension [index] has
  [interpret][array_1(DimSize_[index])][/interpret]
  indexes.

[/loop]
```

Results:

```
array_1 has 3 dimensions:
Dimension 1 has 3 indexes.
Dimension 2 has 4 indexes.
Dimension 3 has 5 indexes.
```

[FORMVARIABLES] CONTEXT

Syntax: [formvariables]text and tags to loop through[/formvariables]

Result: Lists all the form variables and parameters passed to the current page.

Optional Tag Parameters:

- **name=variablename** – The full or partial name of the variables to list.
- **exact=Boolean** – Valid values are “T” or “F” (true or false). If exact=T, then variablename must exactly match (case insensitive) the incoming form variables. If exact=F, then the incoming variable name must only contain the text in variablename. The default value is T.
- **form=form name** – Two possible values are “HTML” and “INCLUDE”. By default, the context returns the form variables sent to the outermost HTML page. If this tag is in an include file and you want to list just those form variables passed to the include file from within the [include] tag, specify “form=include” in the beginning [formvariables] tag.

Optional Context Tags:

- **[name]** – The name of the variable currently looping through.
- **[value]** – The value of the variable.
- **[index]** – A number from 1 to the number of variables, indicating this variable's index in the list.

To display a list of all the form variables available, insert a [FormVariables] context into a template.

For example:

```
The following are all the form variables available to this page:<br>
[FormVariables]
[index],[name],[value]<br>
[/FormVariables]
```

The [FormVariables] context has optional parameters placed within the beginning tag in order to modify the list of form variables produced.

For example:

```
The following are the values of all the form variables with the name "text":<br>
[FormVariables name=text&exact=T]
[index],[value]<br>
[/FormVariables]
```

Listing the variables with a given name is useful for getting the results of a multiple select list or multiple checkboxes with the same name.

For example:

```
[include file=text.tpl&test=Y&category=software]
text.tpl:
[formvariables form=include]
[name],[value];
[/formvariables]
```

The above results in the following output:

```
file,text.tpl:test,Y;category,software;
```

[FREEMEMORY] TAG

Syntax: [FreeMemory]

Placing [FreeMemory] in your template displays the amount of memory available to WebDNA. This number is reduced whenever templates are cached, or databases are opened.

[FUNCTION] CONTEXT

New in 5.0

Syntax: [function name=...&preparse=T/F][function]

Result: This new context enables the WebDNA programmer to call a previously defined block of WebDNA code.

Optional Tag Parameters:

- **name=variablename** – User defined name for the function. The name is then used like a normal WebDNA tag.
- **preparse=T/F** – By default, the WebDNA code that defines the function is stored 'raw' and executed later when the function is called. But if you need to programmatically create the function definition using WebDNA, then you can set 'preparse' to 'T'. This will force the WebDNA engine to first parse the WebDNA in the function definition before storing it for later use.

Our first example creates a function named 'backwards' that will take a variable named 'instring' and display the characters of the string in reverse order. We use the following code...

```
[function name=Backwards]
[text]length=[countchars][instring][countchars][text]
[loop start=[length]&end=1&advance=-1][getchars
  start=[index]&end=[index]][instring][getchars][loop]
[/function]
```

Now the function is defined and stored for later use in the template. To execute the new function, we use...

```
[Backwards instring=abcdef_12345]
```

Function names take precedence over WebDNA global tags. So it is possible to 'override' a WebDNA tag. For example, let's define a function named 'date' that displays the date in bold text.

We use the following code...

```
[function name=Date]
```

```
[ :global:Date ]  
[ /function ]
```

Note that in our function definition, we had to use explicit 'scoping' to access the true WebDNA [date] tag. This is to prevent an infinite recursion when the function code executes, i.e. keeps calling itself. You can learn more about Scope and Scope Resolution in the 'Scope' tutorial.

Now when we use [date], our date function is called, instead of the global [date].

[INCLUDE] TAG

Syntax: [Include file=*FilePath*]

Placing [Include *FilePath*] in your template replaces the [Include] tag with the contents of the specified file. The included file can use any [xxx] tags that will be substituted as though you had typed the entire contents of the file at that place in the template.

Note: Normally all file paths are relative to the local template, or if they begin with "/" they are relative to the web server's virtual host root. As of version 3.0, you may optionally put "^" in front of the file path to indicate the file can be found in a global root folder called "Globals" inside the WebCatalogEngine folder. This global root folder is the same regardless of the virtual host.

Optional Parameters	Description
[INCLUDE file= <i>FilePath</i> &raw=T]	raw=T means the file should be included unchanged, without performing any [xxx] substitutions.
[INCLUDE file= <i>FilePath</i> &fromCache=F]	fromCache=F means a more-recent version of the file should be read from disk, instead of using the cached version in RAM.
[INCLUDE file= <i>FilePath</i> &var1=xx&var2=yy]	Passes any variable names (and their values) you choose into the included template, which can then use [var1] anywhere inside it.
[INCLUDE file=^test.inc]	test.inc is found inside WebCatalogEngine/Globals/ folder

[LASTRANDOM] TAG

Syntax: [LastRandom]

Placing [LastRandom] in your template displays the same value as the last [Random] number displayed. See [RANDOM].

[LISTVARIABLES] CONTEXT

Syntax: [ListVariables *options*]Variable Tags[/ListVariables]

Result: Lists all the text and/or math variables that have been set earlier on a page.

To display a list of all the text or math variables available, put a [ListVariables] context into a template.

Example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

The following are all the text and math variables available to this page:


```
[ListVariables]
[index],[name],[value]<br>
[/ListVariables]
```

The [ListVariables] context has optional parameters that are placed within the beginning tag in order to modify the list of variables produced.

Example:

The following are the values of all the text variables created on this template so far:


```
[ListVariables type=text]
[index],[name],[value]<br>
[/ListVariables]
```

The following are the values of all the math variables created on this template whose variable name begins with "fred":


```
[ListVariables type=math&name=fred]
[index],[name],[value]<br>
[/ListVariables]
```

Listing the variables with a given name is useful for displaying arrays of variables.

Optional Tag Parameters:

The following parameters are optional to the [ListVariables] context:

Optional Tag Parameters	Description
Name	(Optional) The name of the variable to list.
Exact	(Optional) T(ue) or F(alse) whether to exactly match the name of the variable or match any name containing the "name" value. (Default value is true.)
Type	(Optional) Text or Math. Default is to list variables of both types, but if you specify Text then only Text variables will be listed, and if you specify Math then only Math variables will be listed.

Optional Context Tags:

The following tags are available inside a [ListVariables] context:

Optional Context Tags	Description
[Name]	The name of the variable.
[Value]	The value associated with the variable.
[Index]	A number from 1 to the total number of variables, indicating this field's index position in the list.
[Secure]	(available from version 4.0.2rc2 and later) Displays "T" if the variable is secure, "F" otherwise.

[LOOP] CONTEXT

Syntax: [loop start=x&end=y]Any Text or HTML[/loop]

Result: Loops through the enclosing text the specified number of times

Required Tag Parameters:

- **start=x** – The starting index value to begin looping. This value is required and may be positive or negative (depending upon whether you are looping up or down).
- **end=y** – The ending index value when looping. This value is required and may be positive or negative.

Optional Tag Parameters: Advance (The amount to advance by when looping. This parameter is optional and may be positive or negative, the default value is 1.)

Optional Context Tags:

- **[index]** – The current index number between or including Start and End.
- **[break]** – If the [Loop] context sees the [Break] tag while executing a loop, it will stop looping, once it finishes the current loop. Thus the [Break] tag should only appear in a [ShowIf] statement that is evaluated at the end of the loop.

The [Loop] context requires two parameters: Start and End. Looping occurs between (and including) the Start and End value; advancing by 1 each time. You may optionally tell the loop context to advance by a specified amount (other than 1) by using the Advance parameter.

For example (normally you would put the following text into a .tpl file on your server and use a web browser to link to it):

```
[loop start=1&end=[numAdd]&advance=1]
[append db=some.db]...[/append]Record [index] added.<br>
[/loop]
```

Using the Loop context you can easily create a form that adds an arbitrary number of records to a database. The form that links to this page (via a showpage command for example) can specify the data to add for each record and how many records to add ([NumAdd]).

If you chose to add three records, the results would look like this:

Record 1 added.
Record 2 added.
Record 3 added.

For example:

```
[loop start=1&end=7&advance=2][index]/loop]
```

This will loop 4 times using [Index] values 1,3,5,7.

Note: It is possible to loop in either ascending or descending order.

Ordinarily you loop in ascending order:

```
[loop start=1&end=5&advance=1][index]/loop]
```

To loop in descending order, define the beginning of the loop as the greater number, the end as the lesser number and advance as a negative number:

```
[loop start=5&end=1&advance=-1][index]/loop]
```

For example:

```
[loop start=1&end=10]
Index = [index]
[showif [index]=4][break]/showif]
[/loop]
```

The [Loop] context above always stops looping after the fourth loop when it sees the [Break] tag. The tag is placed at the end of the context because it doesn't stop executing until the end of the loop that contains the [Break] tag.

[PLATFORM] TAG

Syntax: [Platform]

Placing [Platform] in your template displays the computer platform (Windows or Macintosh or UNIX) upon which WebDNA is running.

[RANDOM] TAG

Syntax: [Random]

Placing [Random] in your template displays a random number between 1-100. See [LastRandom].

The 'old' Solution

One way that WebDNA programmers have dealt with unwanted return characters, is to wrap line-endings, or other unwanted white space, with WebDNA comments, i.e. [!]/!. So the function definition on the previous page would look like...

```
[function name=add_em_up] [!]
[!]/! [text] result=0 [/text] [!]
[!]/! [loop start=1&end=10] [!]
[!]/! [text] result=[math] [result]+[index] [/math] [/text] [!]
[!]/! [/loop] [!]
[!]/! [result] [!]
[!]/! [/function]
```

Executing the above function, and wrapping the result with URL tags, we get:
"55"

The extra 'garbage' is gone, but using all those [!]/! pairs is cumbersome, and does add some extra parsing overhead.

A Better Solution

The [Return] context can now be used to target exactly what we want the function to return. So our example function now looks like...

```
[function name=add_em_up]
[text] result=0 [/text]
[loop start=1&end=10]
[text] result=[math] [result]+[index] [/math] [/text]
[/loop]
[return] [result] [/return]
[/function]

"[url] [add_em_up] [/url]"
```

Executing the above code, we get: "55"

The extra 'garbage' is gone, and we did not have to use all those [!]/! contexts.

Even if the explicit results of a function call are not significant, for example, when the function assigns the result to some global text variable. It is still a good idea to use the [Return] context in order to cut down on the amount of white space that may be returned to the client browser.

For example:

```
[function name=add_em_up]
```

```

[text]result=0[/text]
[loop start=1&end=10]
[text]result=[math][result]+[index] [/math] [/text]
[/loop]
[text scope=global]result=[result] [/text]
[return] [/return] [!] return nothing [!/]
[/function]

[add_em_up]
result="[result]"

```

Executing the above code, we get:

result="55"

As mentioned in the first page of this tutorial, the [Return] context does not actually 'return' or 'break out' of the function call. So, it is possible to have multiple [Return] contexts in a given function definition. For example:

```

[function name=add_em_up]
[text]result=0[/text]
[loop start=1&end=10]
[text]result=[math][result]+[index] [/math] [/text]
[showif [index]

```

Results in...

"1+2+3+4+5+6+7+8+9+10=55"

The [Return] context is also very useful when creating 'recursive' functions (functions that call them selves until a terminating 'base case' is reached). Here is a sample recursive function that calculates the factorial for a given integer.

```

[function name=factorial]
[showif [num]>1]
[return] [math][num]*[factorial num=[math][num]-
  1[/math]] [/math] [/return]
[/showif]
[hideif [num]>1]
[return]1[/return]
[/hideif]
[/function]

6! = [factorial num=6]

```

The results...

6! = 720

[SCOPE] CONTEXT

New in 5.0

Syntax: [scope name=...]<WebDNA>[/scope]

Result: Enables a WebDNA programmer to explicitly define a block of WebDNA code that has a separate variable space. Meaning that variables defined within the 'scope', only exist for the duration of WebDNA between the 'scope' tags.

Optional Tag Parameters:

- **name** - User defined name for the local variable space. This name can be used with the scope resolution operator to 'access' variables stored in the 'named' variables space (but only for the duration of the scope context).

Named Scope

Lets create a named variable space called 'mytempvars', and create a few text variables in the new scope.

We use the following code...

```
-Scope begin...

[scope name=mytempvars]
[text]a=11[/text]
[text]b=22[/text]
[text]c=33[/text]
List of local scope variables...

[listvariables scope=mytempvars] [name]=[value]
[/listvariables]
-Scope end...

[/scope]

List of global variables...

[listvariables] [name]=[value]
[/listvariables]
```

Result...

```
-Scope begin...
  List of local scope variables...
  a=11
  b=22
  c=33
-Scope end...
List of global variables...
page_number=2
page_name=Scope
edit_link=Scope/Scope_--2---.tpl
back_link=Scope_--1---.tpl
next_link=Scope_--3---.tpl
new_file=Scope/Scope_--3---.tpl
main_title=Scope
```

So you can see that the 'local' scope variables; 'a','b', & 'c', only exist between the [scope] tags.

This is useful when you need to create several temporary variables for a specific block of WebDNA code, but do not want the variables 'cluttering' the global template variable space.

Scope and Functions

WebDNA functions have their own implied scope. Meaning that when you create variables inside of a function definition, the variables are local to that function. The 'name' of the variable space in the function, is the function name itself.

For example...

```
[function name=test_function]
[loop start=1&end=10]
[text]local_[index]=[index][/]text]
[/loop]
[listvariables scope=test_function][name]=[value]
  [/]listvariables]
[/function]

[test_function]
```

Results...

```
local_1=1
```

```
local_2=2
local_3=3
local_4=4
local_5=5
local_6=6
local_7=7
local_8=8
local_9=9
local_10=10
```

Scope Resolution - Step-Wise Method

Because of the addition of 'local' variable name spaces to WebDNA, there will often be occasions when you need to explicitly access variables in a given scope. You can do this using the new 'Scope Resolution' operator: '::'. There are two modes of Scope resolution; 'step-wise' and 'named'.

When parsing WebDNA code, the WebDNA engine will search 'inside-out' for variable matches on a given token, and return the first 'match'. The step-wise scope resolution method can force the engine to 'step over' matching variable names and continue searching for 'outer' variables that match the given token.

This 'step-wise' scope resolution is used as follows...
Any number of 'colon' pairs preceding a variable name.

::]

A usage case...

Often from within an [orderfile] context, you may want to access the global [date] tag. Until now, if you used [date] from within an orderfile context (or on a page called with the 'showcart' command), you would get the orderfile's date, and not the 'global' date value. Now, you can retrieve the 'global' date value using, [::date] from within the orderfile context. Basically the '::' reads as, 'skip the first occurrence of a 'date' value, and retrieve the next (if it exists).'

```
[orderfile file=testcart]
[date]

[::date]

[/orderfile]
```

Results...

```
1/01/2001
03/04/2003
```

You can 'stack' any number of resolution operator '::' pairs to 'skip' to a particular 'outer' instance of a variable...

```
[scope name=scope1]
  [text]a=1[text]
  [scope name=scope2]
    [text]a=11[text]
    [scope name=scope3]
      [text]a=111[text]
      value of 'a' in [scope3] = [a]

      value of 'a' in [scope2] = [::a]

      value of 'a' in [scope1] = [:::a]

    [/scope]
  [/scope]
[/scope]
```

Results...

```
value of 'a' in [scope3] = 111
value of 'a' in [scope2] = 11
value of 'a' in [scope1] = 1
```

Scope Resolution - Named Method

You can directly refer to a particular scope using the scope name between the ':' colons. The 'name' must be qualified by 'named-', i.e.

```
[:named-:]
```

'Reserved' scope names can also be used. In this case you do not need to include the 'named-' prefix, i.e.

```
[:global:]
```

Reserved scope names are discussed on the following page.

Using the orderfile example from the previous page...

```
[orderfile file=testcart]
```

```
[date]

[:global:date]

[/orderfile]
```

Results...

```
1/01/2001
03/04/2003
```

The 'global' name is the reserved scope name for secure template variables.

Using the nested scopes example from the previous page...

```
[scope name=scope1]
  [text]a=1[text]
  [scope name=scope2]
    [text]a=11[text]
    [scope name=scope3]
      [text]a=111[text]
      value of 'a' in [scope3] = [:named-scope3:a]

      value of 'a' in [scope2] = [:named-scope2:a]

      value of 'a' in [scope1] = [:named-scope1:a]

    [/scope]
  [/scope]
[/scope]
```

Results...

```
value of 'a' in [scope3] = 111
value of 'a' in [scope2] = 11
value of 'a' in [scope1] = 1
```

Reserved Scope Names

There are a few 'reserved' scope names:

'global' - refers to the 'normal/secure' template variable space.

'local' - When used inside of a function or scope context, refers to the variable space associated with the current function or scope.

'insecure' - Refers to the 'insecure' template variable space (this space also includes HTML form variables).

A demonstration of named and step-wise scope resolution...

```
[text]abc=123_global[/text]
[text insecure=F]abc=123_insecure[/text]

[scope name=scope1]
[text]abc=123_local[/text]

global 'abc' = [:global:abc] - using reserved 'global' name
global 'abc' = [::abc] - using 'step-wise' scope

insecure 'abc' = [:insecure:abc] - using reserved 'insecure'
name
insecure 'abc' = [::::abc] - using 'step-wise' scope

local 'abc' = [:local:abc] - using reserved 'local' name
local 'abc' = [abc] - using implied scope
local 'abc' = [:named-scope1:abc] - using 'named' scope

[/scope]
```

Results...

```
global 'abc' = 123_global - using reserved 'global' name
global 'abc' = 123_global - using 'step-wise' scope

insecure 'abc' = 123_insecure - using reserved
'insecure' name
insecure 'abc' = 123_insecure - using 'step-wise' scope

local 'abc' = 123_local - using reserved 'local' name
local 'abc' = 123_local - using implied scope
local 'abc' = 123_local - using 'named' scope
```

New 'scope=' parameter

Besides being able to 'resolve' variables in different scopes, you can create variables for a specified scope using the 'scope=' parameter in [text] and [math] assignments. This option will soon be available for other persisted WebDNA object, i.e. Arrays, Tables, XML/XSL objects, etc.

This is useful when you have a block of WebDNA code within a function, or named scope, and need to create/modify a variable in an outer scope.

For example...

Here we simulate a 'pass by reference' by passing the variable name, to receive the results, into a function call.

```
[function name=Backwards]
[text]length=[countchars][in_string][/countchars][/text]
[text scope=global][output_to]=[loop
  start=[length]&end=1&advance=-1][getchars
  start=[index]&end=[index]][in_string][/getchars][/loop][
/text]
[/function]

[Backwards in_string=abcdef_12345&output_to=result]

result = [result]
```

Results...

```
result = 54321_fedcba
```

In the example above, the function definition assumes that the destination variable existed in the 'global' scope. Lets do the same thing as before, but pass in the scope name as well...

```
[function name=Backwards]
[text]length=[countchars][in_string][/countchars][/text]
[text scope=[output_scope]][output_to]=[loop
  start=[length]&end=1&advance=-1][getchars
  start=[index]&end=[index]][in_string][/getchars][/loop][
/text]
[/function]

[!] clear out global 'result' from previous example [!]
[text]result=[/text]
[scope name=scope1]
```

```
[Backwards
  in_string=abcdef_12345&output_to=result&output_scope=scope1]
value of 'result' inside of 'scope1' = '[result]'
[/scope]
```

value of 'result' outside 'scope1' = '[result]' (should be empty).

Results...

```
value of 'result' inside of 'scope1' = '54321_fedcba'
```

value of 'result' outside 'scope1' = " (should be empty).

[SENDMAIL] CONTEXT

Syntax: [sendmail to=address&from=address&subject=text]Email
text[/sendmail]

Result: Sends an email to the specified address.

Required Tag Parameters:

- **to=address** – Recipient of this email, as in address@domain.com.
- **from=address** – Return address, as in youraddress@yourdomain.com.
- **subject=text** – Subject line for email.

Optional Tag Parameters:

- **saveonsuccess** - instructs WebDNA to save or not save the email files after successful transmission.
- **saveonfail** - instructs WebDNA to save or not save the email files after failed transmission.

To send an email, insert a [SendMail] context into a template with the body of the email message inside the context. Specify to, subject, and from information in parameters of the [SendMail] context. WebDNA does not actually send the email; instead it writes a special file into an EmailFolder that the separate EMailer program uses to send the email. If the EMailer program is not running, no emails are sent. The EMailer program does not erase old outgoing email files until it has successfully completed sending the email.

For example:

```
[sendmail to=you@xxx.com&from=me@xxx.com&subject=Hello]
This is the body of the email. The date is [date].
[/sendmail]
```

Any [xxx] values are first substituted for their real values (both in the parameters as well as the body of the message), then the email file is written into the EmailFolder, where the EMailer program checks for new files every few seconds and sends them out.

The EMailer application has a status window listing all emails sent and their status. An error code of 0 (zero) means that the email was sent successfully. If there was a problem sending the email, one of the following error codes displays:

Mailer Error Codes

0	(no error) Mail sent successfully.
100	(netOpenDriverErr)
101	(netOpenStreamErr)
102	(netLostConnectionErr) The above three errors are Open Transport or network errors; if they are transitory, EMailer will retry the message until it is able to send it correctly. If the problem persists, you are likely having other problems and may need to restart your computer or check your network setup.
103	(netDNRErr) Often caused by incorrectly addressed email files; examine the files in your EMailFolder in a text editor and fix or delete them as necessary.
104	(netTruncatedErr) Usually indicates a broken pipe between sender and receiver; message will be resent.
120	(smtpServerErr) Usually caused by a non-responsive or busy mail server; try changing the EMailer Preferences to temporarily point to a known, working SMTP server.
150	(badRecipient) Caused by a bad recipient address; an error will be written to the log and the offending message moved to the EmailCompleted folder.
151	(timeout) EMailer timed out while trying to connect to the mail server and will try again in 30 seconds. Often caused by a busy mail server; if problem persists, try changing the preferences to point to a less busy/more reliable mail server.

Note: If you wish to set custom header values for your email, you can write the mail file yourself using the [WriteFile] context. Emailers file format conforms to UNIX SendMail and Macintosh SIMS/EIMS. See *Email File Format* for more information.

By default, after WebDNA has transmitted an email, the email file is moved from the EMAILFolder to either the EMAILCompleted or EMAILProblems folder, depending on the success or failure of the email transmission. You can use the 'SaveOnSuccess' and 'SaveOnFail' [SendMail] parameters to instruct WebDNA NOT to 'save' the email files. Here is an example:

```
[sendmail
  to=steve@here.com&from=rad@there.com&subject=WebDNA&Save
  OnSuccess=F&SaveOnFail=T]
WebDNA leads the way to productivity.
[/sendmail]
```

In the example above, the email file would not be 'saved' to the EmailCompleted folder after a successful transmission, but it would be saved to the EmailProblems folder if there was an error in transmission.

HEADER FIELDS

You may set any shopping cart header field (such as Name, taxRate, Address1, etc.) at the same time you add a product to the cart. See *SetHeader*.

To change a line item in a visitor's shopping cart, insert a SetLineItem context into the template (alternately, you may use the ShowCart command from a URL or a FORM). Whenever WebDNA encounters a SetLineItem context, it opens the shopping cart file and changes values in a line item (identified by its index). The item's quantity, textA-E, and cart header fields are all changeable. You can use a different price by creating a Formulas.db database. Also see Remove, Clear, ShowCart, [AddLineItem] and Purchase.

For example:

```
[setlineitem cart=5678&index=3&db=catalog.txt]quantity=4
&textA=Blue[/SetLineItem]
```

Shopping cart file "5678" is opened, and line item 3's quantity is changed to 4 and textA changed to "Blue" (as specified in the context above).

[THISURL] TAG

Syntax: [ThisFile]

Placing [ThisFile] in your template displays the platform-specific file system full path of the current template page being displayed. The path separators are platform-specific, so the path will look different depending on what MacOS, Windows, or UNIX computer system is used.

[VERSION] TAG

Syntax: [Version]

Placing [Version] in your template displays the version of the currently running WebDNA program or plug-in.

[!] COMMENT CONTEXT

Syntax: [!]Any Text to be Hidden[!]

Description: Hides text, usually a comment for the WebDNA developer to read.

To hide comments in HTML pages so you can see them when you are developing a template, but web visitors cannot see them, put them inside a comment context. Anything inside the [!] context is hidden, and WebDNA will not execute.

For example, normally you would put the following text into a .tpl file on your server and use a web browser to link to it:

```
[!]This text will be hidden from web visitors[!]
```

Many times you will want to keep notes for yourself in a WebDNA template, yet you do not want outsiders to see these notes.

Unlike HTML comments, which are sent to the browser even though they are hidden from view (and can be seen with a View Source command), WebDNA comments never get sent to the remote browser. This shortens download times, and provides a degree of protection against prying eyes.

Using WebDNA Tags

WebDNA replaces tags with their computed value. Since they are not containers, there is no ending tag (`[/xxx]`) required. Some tags may be placed in any template processed by WebDNA; others may only be used if they are within the specified context. Additionally, if you have defined a SUFFIX-MAPPING and ACTION for WebDNA to process `.tpl` or `.HTML` pages, then using a web browser to link to any `.tpl` or `.HTML` page on your site is equivalent to performing a ShowPage using that file as a template.

Depending on your Preference settings, you may have to include the `<!--HAS_WEBDNA_TAGS-->` statement at the top of your files to tell WebDNA to interpret the `[xxx]` tags inside the file.

PREFERENCES

Depending on the preference settings you defined in the WebDNA Administration template, you may need to include `<!--HAS_WEBDNA_TAGS-->` at the top of your files to indicate that WebDNA should interpret the `[xxx]` tags inside the file. This tag is case sensitive. `<!--HAS_WEBDNA_TAGS_XML-->` indicates that the WebDNA on this page conforms to the newer XML-style syntax.

PARAMETERS

Some tags do not require any parameters, some tags require only one parameter, and others require multiple parameters. Tags that use a single parameter often do not require the name of the parameter to be used in the tag (although we do not recommend this; it's simpler and less error-prone to always use the named parameter everywhere). Tags that require multiple parameters must have them specified in name/value pairs with the ampersand character “&” connecting multiple parameters. The names of the required or optional parameters are not case sensitive, but must appear exactly as written, without extra spaces. Quotation marks should not be used to surround the value of a parameter as is the case with many HTML values (unless a quotation mark is part of the value itself).

Many tags take optional parameters. Either the parameters default to a standard value or they are user definable.

Examples:

No parameters:

```
[browsername]
```

Single require parameter, name not required:

```
[include filepath]
```

Multiple parameters, name/value pairs separated by “&”:

```
[include file=filepath&raw=T]
```

Optional, user definable parameters. The user defines both the name and value of the parameter. Parameter “name” is passed to the included file, but not required by WebDNA:

```
[include file=filepath&myname=myvalue]
```

ITALIC TEXT

The name of a parameter is fixed, but the value assigned to it can vary. All text that is italicized refers to values that should be replaced with your appropriate data.

Example:

```
[include file=filepath]
```

The single required parameter, “file” must appear exactly as shown. The actual path, however, varies depending upon your situation. The value *filepath* should be replaced with the actual file path used.

PATHS

All tags with parameters specifying paths to databases, templates, or files should be specified in standard URL style. That is, relative paths from the current location begin with the file or folder name. On Macintosh and UNIX systems, full paths from the root of the virtual web server folder begin with a forward slash “/” character. On Windows, full paths that begin with a forward slash “/” are from the WebCatalogEngine CGI folder (where DBServer.exe resides). Folder names are separated with the forward slash character as well.

Example:

```
[include file=MyFile/header.inc]
```

Path is relative from the current location. The included file is found in a folder called "MyFile" inside the folder that contains the template.

```
[include file=/Includes/header.inc]
```

Path is global, no matter where the template is located. The file is found in a folder called "Includes" in the root web server folder.

FORM VARIABLES

Syntax: *[name]*

The variables or parameters of a form can be included on a page that the form links to by enclosing the name of the variable in square brackets. Since the user determines the name of a form variable, the actual name and associated value, will vary.

Example:

```
<a href="test.tpl?var1=Hello&var2=World">Hello</a>
```

- or -

```
<form method=post action="test.tpl">  
<input name="var1" value="Hello">  
<input name="var2" value="World">  
</form>
```

Both of the preceding forms (first method=get, second method=post) link to a page named test.tpl.

If test.tpl is the following:

```
The value of var1 is [var1]<br>  
The value of var2 is [var2]<br>
```

the text returned from this example would be:

```
The value of var1 is Hello  
The value of var2 is World
```

Using WebDNA Contexts

A context is the WebDNA term for enclosing tags that have a beginning and ending form and for enclosing a portion of text. The Tags defined in the first part of the WebDNA reference did not require closing tags. An important characteristic of contexts is that they may be nested, that is to say, one context can enclose another. In fact, some contexts can only be enclosed

within certain other contexts. Usually it is obvious whether a context must be enclosed within another context. For example, it would make no sense to have a [founditems] . . . [/founditems] context outside of a [search] . . . [/search context]. Unless you search for items, there can be no found items. In many cases, the actual result returned by a specific context may depend upon its being within an enclosing context.

Obviously, you can include tags inside a context as well. In fact, you can use WebDNA tags as parameter values within the beginning context tag itself.

A context has two kinds of parameters. Both kinds of parameters are present in the [AddLineItems] context described first. In addition, it is possible to have special tags that are only available within a certain context.

TAG PARAMETERS

A tag parameter is a parameter that is found within the beginning context tag. This is very similar to the tag parameters mentioned in the previous section.

For example:

```
[search db=databasepath&search criteria]...[/search]
```

CONTEXT PARAMETERS

Context parameters are found between the enclosing context tags (as opposed to within the tags themselves. Not all contexts have context parameters (or tag parameters for that matter). Many contexts allow text of any kind between the beginning and ending tag. Those that require context parameters require that the parameters be specified in the same way they are for tags.

For example:

```
[append ...]sku=001&description=productName[/append]
```

CONTEXT VARIABLES

Within a context, you can access specific variables that don't function outside of the context. This is especially true for any context that loops through values as in the Founditems context that loops through found items. At the very least, a looping context should have tags like [Index] available to let you know which item you are currently looping through. Other tags make sense

within their context alone. The following two need to be recopied wherever they came from...

For example:

```
[loop start=1&end=2]Current Index: [index][!/loop]
Context: ! - (Comment)
Syntax:
[!]some text[!]
```

Result:

Use the Comment context (the exclamation point) to add text to your HTML/WebDNA pages that is removed before being sent to the browser.

To add additional line breaks or text comments, without your visitors seeing the text when viewing the source of their Web page, enclosed the text in the Comment context.

For example:

```
[!]Here's where we add a record to the database[!]
[append db=...][!/append]
```

Using WebDNA Commands

WebDNA commands can be sent to WebDNA in essentially two ways: 1) when they are included within the URL of a hypertext link (an <A HREF> tag), and 2) as values in a hidden form field.

For example:

In the Tutorial, users enter the shopping site from the first page sent from the TeaRoom site by clicking on the <A HREF> hypertext link that contains the search command:

```
<A HREF="Search.tpl?command=search&geSKUdata=0&db=TeaRoom.db
&categorysumm=t&categorysort=t&max=50"> Click HERE to Enter the
store.</A>
```

When the visitor clicks on the hypertext link, the browser sends the URL to a web server. The URL requests the server to send it back the page "Search.tpl". The server receiving the command has been configured to recognize that the .tpl suffix requires that page be processed through the WebCatalog CGI or plug-in. In this case, the command that will be executed

by WebDNA is the Search command. WebDNA searches the TeaRoom.db database for all the records and summarizes them in a list by category.

We could send the same command using an HTML <FORM> tag.

For example:

```
<FORM METHOD="POST" ACTION="Search.tpl">  
<INPUT TYPE="SUBMIT" NAME="command" value="search">  
<INPUT TYPE="SUBMIT" VALUE="Summarize by Product Categories">  
</FORM>
```

When the visitor clicks on the submit button the same information is sent to the server and WebDNA that was sent by clicking on the hypertext link.

COMMAND= NOTATION

You must specify the command as a parameter after the question mark. This style is used for Windows web servers and provide cross platform compatibility for your templates. When using this style, the URL is followed immediately by a question mark and then the list of parameters. A parameter whose name is "command" and has a value from the commands listed in this section is used when executing the template specified.

Most examples use this notation in order to provide compatibility with both Windows and Macintosh.

For example:

```
<a href="test/text.tpl?command=search&lefield1datarq=2">
```


Chapter 4 – Advanced Uses of WebDNA

WebDNA is a flexible dynamic responding text engine. It can respond to a request from multiple users to provide information back to the Web Server that satisfies each browser request. An example of this is the multi-user day planner feature. Users that have different meeting agendas can be presented with their particular calendar of meetings and events following a request to see their day planner.

This chapter describes how some ideas for WebDNA can be used dynamically to provide multiple services and strategic business solutions. Further, once you come up with these solutions you can encrypt them so that others can only use them and not see how you are doing it.

Encrypting Templates

In addition to encrypting text using WebDNA's [encrypt] context, you can encrypt entire WebDNA templates so they can be distributed and used without having the raw WebDNA in the templates visible (i.e. such as the contents of an email). This is very useful for developers wishing to sell WebDNA-based solutions.

Sometimes you will want to give someone else a WebDNA template without letting him or her see your "source code" in the file. For example, if you create a WebDNA solution that is for sale, you may not want others to be able to read the templates and make modifications or see your proprietary algorithms.

HOW TO ENCRYPT TEMPLATES

To create an encrypted template, you must first design and debug the template as you normally would. Then use the [Encrypt] context with a seed of your choosing to create the encrypted version of the original template. Note that you shouldn't make your seed value public in any way. After the templates have been encrypted, a special tag must be added to the top of the file so it can be recognized as an encrypted file and then your seed value must be encrypted using WebDNA's default encryption.

Use the following WebDNA to encrypt your file and add the special tag to the beginning of the file. Note that you may put any text, copyright information for example, before the header tag. The encrypted WebDNA must start on the line after the header tag. For example, in order to use WebDNA to automatically convert your template (file.tpl) to an encrypted template (newfile.tpl) you would execute the following:

```
[writefile file=newfile.tpl&secure=F]Copyright © 2009 WebDNA Software Corp.  
http://www.webdna.us/  
<!--HAS_WEBDNA_TAGS[!][!/_ENCRYPTED_2-->  
[encrypt]seed=XXXXXXX&product=WCAT[/encrypt]  
[encrypt seed=XXXXXXX][include file=file.tpl&raw=T][[/encrypt]][/writefile]
```

After using the above template, your hard disk will contain a file called “newfile.tpl” which you can give to other users, confident that they cannot read or modify it. Using the template is easy—it works just like any other template.

Note: [!][!] is a special trick to fool WebDNA into thinking this page is not an encrypted page, and should be treated like a normal template. The [!][!] is essentially removed during processing, which causes the resulting template to contain the now-correct tag that indicates it is encrypted.

ENCRYPTING THE HEADER TAG

The following header tag is valid for recognizing encrypted files:

```
HAS_WEBDNA_TAGS_ENCRYPTED_2
```

In addition to encrypting your seed value, you can specify which product this template works with. Use “WCAT” for WebDNA, “TYPH” for Typhoon, or “WDNA” for any WebDNA product.

Link to an encrypted page and WebDNA automatically decrypts and processes the page. In order to prevent someone from displaying or accessing the decrypted templates, the following precautions have been made:

- . command=raw – Does not work with encrypted templates; nothing is returned.
- . [include raw=T] – Cannot be done on encrypted templates; nothing is returned.

Talk List Subscription and Archive

WebDNA provides several ways for users and developers to share their experiences, concerns, comments, and knowledge in mastering the topic they are interested in. Visit the WebDNA Development Resource Center located at <http://dev.webdna.us>.

The main page provides news and announcements along with any current beta versions of the software. You can download fully functioning demonstration copies of WebDNA and obtain trial serial numbers.

The links along the top of the page provide access to the discussion forum, white papers, the latest versions of WebDNA for download, a reference to the WebDNA Programming Language, Video Training, a Talk List with searchable message archive and more.

To use the Forum, click the Register link on your first visit and respond to the confirming email. After that, you can log on and post your messages and questions using your username and password.

The Talk List Archive provides years of questions and answers by other programmers. In addition to being able to usually find the answer you need, you can subscribe to our active talk list where you will usually receive answers from experienced users quickly. The talk list is one of the most active in the industry and one of the most valuable parts of WebDNA.

Note: Only messages under 10K will be retrieved.

Mail Server:

Username:

Password:

Delete email after checking

[Powered By WebCatalog](#)

Figure 12. Shared POP Mailbox Setup Screen

For further information, look at the WebDNA in the example.

Generating Online Banner Ads

Through the use of WebDNA, you can generate online banner advertisements and track them. The characteristics of the banner ad can include such things as placement and size, frame border, refresh frequency, content links for different advertisers subscribing to the service, and ad statistics reporting and email functions. From the Welcome to WebDNA page, you can see an example of a banner ad under the WebDNA Community List link.

There are other advanced uses of WebDNA, so many in fact, that to put them all in this reference manual would be a manual within itself. However, another notable advanced usage would be Sales Contact management. For a robust web server used by a sales organization, you might need to design a template that fulfills a multi-user, multi-access need according to the team hierarchy. In addition, you might need to make available an array of

interfaces to serve various remote computing clients such as a Palm Pilot or a notebook computer using Mac or Windows. The interface issues are a part from the individuals that would have access to view and/or provide Sales data from the field.

Dreamweaver Integration

WebDNA includes WebDNA extensions that can be added to the Dreamweaver web development program. With these extensions, WebDNA commands can be called within the html used to develop the site's web pages. Refer to the Dreamweaver section in Chapter 4 of the WebDNA User Guide for a description of how WebDNA can be added to Dreamweaver web page design and layout.

XML Syntax Explanation

To give everyone a head start on playing with the new-style syntax, here's a primer. We are calling it XML syntax only because it is not HTML and it tends to follow the guidelines of XML and it looks a lot like XML. It is basically just the type of syntax that most graphical editors like Dreamweaver tend to expect, and **not** chew up like the original [classic] syntax.

The general rule is to substitute "[" with "<DNA_", "]" with ">", and named quoted-value pairs instead of ampersand-delimited pairs. You can see a the new syntax in the included TeaRoomXML example, which has been ported to XML syntax. So the following classic syntax:

```
<!--HAS_WEBDNA_TAGS-->
[loop start=1&end=10]
[index]
[/loop]
```

becomes

```
<!--HAS_WEBDNA_TAGS_XML-->
<DNA_loop start="1" end="10">
<DNA_index>
</DNA_loop>
```

All parameters **must** be named, and **must** be quoted, even if they are numeric or have no spaces in them (also a rule of XML). At some point we may even go so far as to require lowercase tag and parameter names, as XML does, in order to help out with future XML editors.

Certain contexts such as [ShowIf] are required to have a named parameter, so they become:

```
<DNA_ShowIf expr="12<13">...</DNA_ShowIf>
```

You can't get away with unnamed parameters for [Include] either:

```
<DNA_Include file="fred">
```

Again, all of these named things may look funny when you type them out, but they help products like Dreamweaver immensely.

In order to make it easier for you to port sites one-page-at-a-time, we have gone to great lengths to let you intermix the syntax. [include], for instance, automatically assumes the file you're including will be the same syntax as the template you're including *from*.

```
<!--HAS_WEBDNA_TAGS--> means classic syntax for this whole page
<!--HAS_WEBDNA_TAGS_XML--> means XML syntax for this whole page
[include file=fred.inc] will parse fred.inc using classic syntax
[include file=fred.inc&xmlSyntax=T] will parse fred.inc using XML syntax
<DNA_Include file="fred.inc"> will parse fred.inc using XML syntax
<DNA_Include file="fred.inc" xmlSyntax="F">will parse fred.inc using classic
syntax
```

Be careful with quotes! It is easy to forget and write something like [include file="fred.inc"] <-- those quotes are literal in classic syntax, and will create wrong results, where it will look for the literal quote in the filename.

Embedded quotes need to be escaped, also to help with Dreamweaver parsers:

```
<DNA_ShowIf expr="the letter \"a\" ! \"b\">">
```

The idea here is that expr="...a bunch of stuff..." must know when it's hit the ending quote. So any embedded quotes you put inside have to be preceded by \, just like you do in JavaScript.

Arrays: the old trick of using [interpret][array[index]][/interpret] still works in XML syntax, but we had to do some fancy footwork to recognize it.

The syntax looks like this:

```
<DNA_Interpret><DNA_array<DNA_Index>></DNA_Interpret>
```

Undefined variables: classic syntax still treats undefined [x] as literally "[x]", in order to handle the case where you're just typing some plain text on a page that happens to have brackets around it. We did this because we have no way of knowing when you mean literal text and when you mean WebDNA variables. But under XML syntax, we know for certain that you are trying to

get at a variable, because there is no ambiguity about being a WebDNA construct. So in that case we output "undefined" in its place.

Security

Security to the content provided on web pages utilizing WebDNA is dealt with on a per template basis, as well as creative uses of [ShowIf] around information within a template. You can use WebDNA's [Protect] tag to name any user group from the Security section under the Administrator section of the [Administration](#) link. The tag itself can be positioned anywhere on the template where it is to be used. For any template, use of the protect tag causes the web server to collect the user name and password to challenge the user to access the template. Protect causes the user name and password to be used to look up the account in the WebDNA User.db – the groups they are members of are used to compare against the groups allowed to access this the template. The protect tag is an all or nothing access to a template, you can further refine your protection of data by using [showif] contexts within the template.

All protected templates will use the current username and password sent by the browser and compare that against the system users, those in users.db, administered by the security link in your WebDNA admin. Should the user and password not match a user that is a member of the groups that are being allowed access to this template, a challenge for authorization will be returned. In StoreBuilder, the protect tag is used to allow the Admin group as well as the store administrator group control over the store – the store admin group is a preference in the merchant screen for StoreBuilder.

The function of comparing the passed in username and password against the users.db is done from within a special template called MultiGroupChecker. Within MultiGroupChecker straightforward WebDNA is used to perform the logic and you are welcome to extend this to fit your business needs, such as, the acceptable range of IP values, store access hours, and other levels of security as needed. Keep in mind that any changes to this special template effect the entire server.

To further aid security for the store, WebDNA 4.0 has added an optional expiration date for each account. The expiration date is set on the Security form in the WebDNA administration. When the expiration date is passed, the user's access is treated as if it doesn't exist.

MACINTOSH WEBDNA SECURITY NOTES

General Assumptions

Physical Security

- . It is assumed that your web site is physically secured in such a way that unauthorized persons cannot read/write files to its hard disk. This includes people physically standing at the box, as well as remote file read/write such as the following:
 - . FTP
 - . File Sharing
 - . SiteEdit Pro is not a problem because admin password has always been required to display WWWOmega files
 - . Timbuktu
 - . PCAnywhere
- . Obviously you've got big problems if unauthorized people can read/write your hard drive at will. All of these products require passwords, so they are OK to use as long as the password is not compromised.

Secure CGIs

- . It is assumed that no third-party CGI/Plugin installed on your web site will be allowed anonymous users to display, erase, or modify any of WebDNA's files which are marked with WWWOmega filetype. The Crack-a-Mac contest discovered a problem with Lasso that allowed it to read a SiteEdit Pro password file that would have been secure otherwise.
- . Older versions of Lasso (now fixed)

- . NetForms
- . SiteEdit Pro is not a problem because admin password has always been required to display WWWOmega files.

Secure Web Server

- . It is assumed that your web server software itself will never display files that are marked with WWWOmega filetype. Unfortunately at this time only WebSTAR does this correctly. All other servers present a risk.
- . For other servers, we suggest you create a “WebDNA” realm that is password-protected so that visitors cannot display URLs leading to files inside the WebCatalogEngine folder itself (where Users.db password database is stored).

Password Security

- . It is assumed that the passwords to your system are secure and that unauthorized people do not know what the passwords are.
- . Many web protocols make no attempt to encrypt passwords, so for instance FTP and even HTTP realm passwords are sent clear-text across the net. You are vulnerable to packet-sniffing attacks if you use these protocols.

Different Kinds of Attacks

Guestbook

- . **Problem:** Anonymous visitor puts WebDNA into their guestbook comments, and then ‘executes’ the guestbook.db file as though it were a template.
- . **Solution:** Change filetype of guestbook.db to WWWOmega so WebDNA won’t execute it. Make sure preferences don’t allow “.db” as a template suffix.

Physical

- . **Problem:** hopefully this is obvious—people can erase/modify/view any file on your hard disk. All credit cards and passwords are freely available to anyone who has physical (or FTP) access to your server. They can also steal your computer.
- . **Solution:** don't allow unauthorized people into your server room. Don't give FTP password access to sensitive files (such as Users.db, SiteEdit Passwords, etc). Change passwords frequently.

Viewing WWWOmega Files

- . **Problem:** Many programs assume that if they mark their files with WWWOmega filetype, then those files will never be displayed to an anonymous outsider. WebSTAR, WebDNA, and SiteEdit Pro honor this security feature. Other web servers/CGIs (such as Quid Pro Quo, NetForms, Lasso) will display these files, and as such can cause a security breach. WebDNA's sensitive files, such as Users.db and Order files containing credit card numbers, are marked with this special code—but if you have any other CGI or web server installed, they may allow remote viewing of that file.
- . **Solutions:** Install the latest versions of Lasso, NetForms, etc. Add a realm password for "WebDNA" so that outsiders cannot display URLs inside your WebCatalog folder. Move your WebCatalog folder outside the web server hierarchy, and give it an unusual name.

Change Preferences/Passwords Remotely

- . **Problem:** Outsiders may try to use WebDNA's remote administration features to change your preferences.
- . **Solution:** Do not give out any of your administration passwords. WebDNA protects its preferences and Users databases as 'special cases'—it does not allow modifications without a proper admin username/password.
- . **Problem:** Outsiders upload a new WebCatalog Prefs file via FTP or file sharing.

- **Solution:** Do not let outsiders have access to WebDNA's folder. Do not give out your FTP passwords.

Denial of Service

- **Problem:** Mischievous visitors want to prevent others from accessing your site, so they flood your site with requests, thus causing your server to bog down.
- **Solution:** Use your web server log to backtrack the IP address of the offender, and ask their ISP to discontinue their account. Other than that, there is not much you can do, other than to write some custom WebDNA that senses the IP address and sends warning messages to the user.

Too Many Returned Records

- **Problem:** A variation of denial-of-service attack, this makes use of the fact that one can set the number of returned records from a search via a remote parameter (max=50) in the search URL. This can cause huge amounts of data to be returned for each search.
- **Solution:** Either build all of your search templates to use embedded search contexts (which will never look at the URL value of max), or set your WebDNA Preference for max returned records to a smaller number.

Unintended Database

- **Problem:** Outsider creates a 'homebrew' URL that performs a search using a different database file than you intended. For instance, the search results template provided in GeneralStore could be used to display information from a database other than the GeneralStore's catalog.txt file.
- **Solutions:** Never create a template that contains [username] or [password] in its [FoundItems] loop unless that template is also protected with [Protect Admin]. Make sure your WebDNA preferences only allow ".db" and ".txt" as database extensions. Always use

embedded [Search] contexts so that outsiders cannot change the db=xxx parameter.

Unintended Template

- . **Problem:** Outsider creates a URL that leads to a file (such as your Users.db file) that is not a template, and displaying that file will either reveal sensitive information or cause malicious WebDNA to execute.
- . **Solution:** WebDNA will not display WWWOmega files. Make sure your databases have this filetype. Make sure "Require HAS_WEBDNA_TAGS" preference is turned on. By default, WebDNA's [WriteFile] context creates files with WWWOmega filetype.

Packet Sniffing

- . **Problem:** Outsider with sophisticated packet sniffing software can see your realm passwords and FTP passwords as they go by on the net.
- . **Solution:** This problem extends far beyond WebDNA, but is rare because of the 'wiretap' difficult nature of the attack. Perhaps the only solution today is to make sure you only access your sensitive data via SSL.

Hierarchy

- . **Problem:** Outsider sends \$ShowPage command that specifies a file outside your web server's hierarchy.
- . **Solution:** WebDNA prevents attempts to step outside the web file hierarchy, but newer servers with multi-homing ability may someday circumvent this. Aliases to folders also make it more difficult to maintain a clear security map. To help with this, just make sure your WebDNA preferences will only allow ".TMPL" suffixes for templates—which alone should prevent outsiders from viewing sensitive files.

Aliases

- . **Problem:** Unwanted folders become visible to outsiders. Creating an alias to a folder outside your normal web hierarchy can open such folders to outside viewing. WebDNA automatically resolves aliases to

folders (and files) in its template parameters as though they were part of a standard URL path.

- . **Solution:** Be aware that folder aliases allow outsiders to access any files in the resolved folder (only if the person knows the filename). Do not create such aliases unless you intend for all the files in that folder to be accessible.

AppleScript

- . **Problem:** Mischievous webmasters can create WebDNA templates that contain unwanted AppleScript commands. WebDNA executes any AppleScript placed inside its [AppleScript] context. Anyone who has the ability to create or modify template files on your server can write any AppleScript program. Since AppleScript has no security provisions, it is possible to create scripts that erase the entire hard disk, access sensitive information, or even change passwords.
- . **Solution:** Do not allow un-trusted people to create or modify template files on your hard disk. Future versions of WebDNA may provide a preference that disables AppleScript entirely (or forces it to be Admin password-protected).

Command=Raw Views Sensitive Files

- . **Problem:** WebDNA's Raw command allows outsiders to view the 'raw' WebDNA of any template. Templates can contain sensitive information such as passwords (for the [Authenticate] tag) and proprietary WebDNA.
- . **Solution:** The Raw command is protected with Admin-level passwords. Do not give out any Admin passwords to un-trusted persons.

Mischievous Webmaster / Store Owner

- . **Problem:** WebMasters or store owners who have the ability to create/modify files in their own WebDNA folders can write WebDNA templates that display sensitive information. Because WebDNA assumes that all templates are 'trusted' (meaning that no outsiders are

allowed to create files on your hard disk), it allows WebDNA to access any database in any folder—even ones from other 'storefronts' on your computer. Order files with credit card numbers are also easily displayed.

- . **Solution:** Do not allow un-trusted people to create/modify WebDNA templates on your hard disk. Similar to [AppleScript] issues.
- . **Commerce:** Price/Shipping/AccountNum Attack
- . **Commerce:** Stolen Credit Card
- . **Commerce:** Bad Credit Card Number

AREAS TO WATCH FOR SECURITY THREATS

- . Orders Folder
- . CompletedOrders Folder
- . EMail Folder

Uploading Files

In WebDNA, upload is not a single context. Inside of the StoreBuilder, there are two files you should look at:

[/admin/upload_file@.tpl](#)

Also, the file:

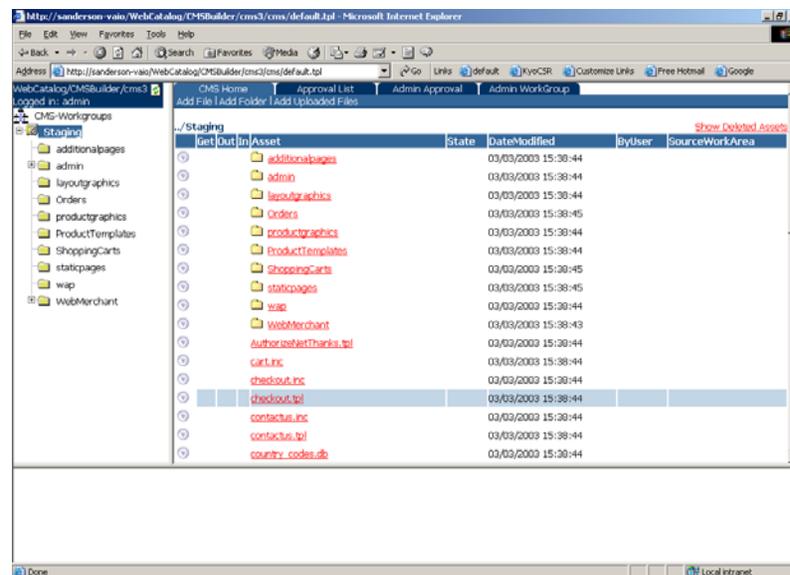
/admin/upload_file.tpl is the one that displays the interface to the user and sets up the variables for [upload_file@.tpl](#) as the destination. In this way, the interface is reusable so that you can copy the two files wherever you wish and pass the right parameters.

WebDNA Content Management System

WHAT IS WEBDNA CMS?

WebDNA Enterprise CMS is a client-server solution that will aggregate three well-known patterns: CMS (Content Management System), RCS (Revision Control System), and CVS (Concurrent Versions System). For the WebDNA Enterprise CMS system, the functionality involved with these three different aspects will be inseparable.

Development teams (“workgroups”) are created and individual users gain membership to one or more of these teams. The concept of privilege-based roles applies to team membership, meaning that given a context workgroup, an individual user is assigned one or more roles. Privileges are assigned statically to an individual role.



First, the authentication/privileges mechanism is the functions as the entry point to consuming system functionality. A given user must first authenticate prior to using the system. All content-manipulation/functionality will be initiated and invoked via an intuitive GUI-based client, based on the familiar file-browser “explorer” pattern.

Content Management System Component

The CMS aspect will allow users within a given development workgroup to manage and organize their individual document/project development efforts based on the familiar file/directory-structure paradigm. Depending on the privileges assigned to the user, the user may, for example, create new, move, and delete files and directories (content).

Revision Control System Component

The RCS aspect will track modifications made to content based on the familiar check-out/modify/check-in paradigm. Given that modification history will be persisted, the system will also allow rolling back to previous versions should the need arise.

Concurrent Versions System Component

The CVS aspect is based on exclusive-locks. As such, it will track file-locks and modifications in other workgroup's work areas so that two users will not be allowed to have the same content (but in a different work area) locked concurrently. All modifications are submitted from a given workgroup's work area to a staging area, simply referred to as "Staging". The revision/version in Staging then becomes the master against which potentially disparate versions in other workgroup's work areas are compared.

Optionally, an admin (workgroup or sysadmin) can specify a structured approval workflow/chain, indicating that various users must flag the request as either approved or denied. In general, a workflow is a map of a set of approval checkpoints. An approval "checkpoint" is simply a combination of a workgroup user and his approval/denial. The actual approval checkpoint map can vary but overall it is a collective indication of whether or not a given submission request gets processed.

USING WEBDNA CMS

Note: There is a right-button context menu that works in both the tree view and list view. For MAC users with only one button, there is a 'menu' icon in the list view that pops up the same thing.

User Types

There are five 'roles' users can assume:

- **WAdmin** – This privilege provides administrative privileges for a given workgroup.

- **Editor** – This privilege can edit content of a given workgroup, but you cannot create new content or delete content.
- **Author** – This privilege can edit, create, or delete content in a given workgroup.
- **Viewer** – This privilege has read only access.
- **Unauthorized** – This privilege provides no access. It is useful for disabling access for a user who should not be fully deleted from the system.

Every user can see the WorkGroup Admin tab. This is done to allow users to see who belongs to a given workgroup. However, only a WGAdmin for that particular workgroup can change the settings for users.

Approval Groups

The WGAdmin for a particular workgroup can establish the approval groups. Once an approval group is created, the staging admin can then 'attach' an approval group to any particular asset within staging's files or folders. Any time an asset is checked in, the check in process searches recursively from that 'leaf' asset up the hierarchy until it finds an approval group.

For example: If you want one approval group for all of staging, you attach it to the root staging folder. If you want a particular subfolder to have a different approval group, you attach it to that subfolder. Setting a different approval group for a subfolder will allow the recursive search to be encountered first before any approval group 'above' it.

If you have an approval group assigned to a folder, and you have one file that you DO NOT want any approval required ... create an 'empty' approval group and attach it to that one file. It will be encountered first before the approval group attached to the folder containing the file.

Staging Content

The 'staging' workgroup is the repository for all file change history. If you add new files or folders in your respective workgroups, they have to be 'checked-in' to become part of the staging repository. Likewise, if you delete a file or folder, it needs to be checked-in for that change to affect staging. If there is an approval workflow defined, the deletion has to be approved before it is applied.

Deploying Content

There is a 'deploy' action under the Workgroup Admin tab, which launches a separate dialog window for specifying a local production deployment path relative to the file path of the staging folder for that CMS instance. Using the relative path option allows the user to select any local path to copy the staged files to. Once this option is executed, the deployed files will be listed.

Since deploying folder and files to production could be local or remote operations, or involve any number of 'custom' issues for a given customer, this one template will be unencrypted so the user can create their own custom WebDNA to automate the deployment from the staging area into production.

Note: When deploying files to a production environment, it is necessary to flush the databases in the production environment prior to migrating the staged files. This will prevent databases that are cached in memory from overwriting newly migrated databases.

Uploading Multiple Files

There is a mechanism to "add a file" and to "upload" a file over an existing file (if it is already checked out). For 'bulk' uploads of numerous files, there is an 'upload' folder under the CMS instance directory where a user can perform bulk FTPs of files. After the bulk upload, the user can use the "upload files" action, which will then import all of the files from that folder into the CMS system.

Note For Uploading Binary Files

Binary files are editable in the CMS, but will be treated as ASCII-type in the editor. The only way to update or get new versions of a binary file is to check out the file from Staging and then right-mouse click to use the Upload function. This function will update and replace the file in the CMS instance (if it has been checked out), so that the most recent version can be checked back in to Staging.

Advanced WebMerchant Topics

WebMerchant is built as an open source product. It is fully modifiable and not limited to only near real time processing. For example, for those internet payment methods that only support real time debit or credit purchase transactions, a direct link between the customer and the web payment

merchant can occur while the shopping cart transaction is held off to the side until the Internet cash process completes.

When a transaction payment method authorization results in a bad order, WebMerchant can send an email to alert the store clerk to call the customer to find out what the problem is in following through with the order. When a transaction payment method authorization results in a good order, WebDNA logic can be used to automate SKU quantity-on-hand decrementing to reflect the adjusted stock on hand for an item. The adjustment may be derived from the import of a Quickbooks (or other accounting software package) file once per day.

USING ACCOUNT AUTHORIZER

There are times when you wish to receive payment for items without using a credit card—perhaps your company provides customers with some kind of purchase order authorization or special “On Account” payment. You can verify payment in this case by making changes to the AccountAuthorizer.inc template file. The sample file provided will authorize account number 4000300020001000 as “good” and any other value as “bad”. Changing the WebDNA in AccountAuthorizer.inc to your own custom code (perhaps a lookup into a database of known good account numbers) gives you the flexibility to design your own payment scheme. WebMerchant only uses this scheme when the “payMethod” field in the order file is set to “AC”, as opposed to “CC” for credit card payments.

Note: You can use both types of payment methods and direct WebDNA to set the payMethod field accordingly when the order payment method is read.

USE OF EXTERNAL ACCOUNTING SOFTWARE WITH ORDER FILES

You can use different accounting software packages to import order file data from WebMerchant. For example, in the Quickbooks or Manage your own Books software product, you can take the name of the WebMerchant order file as an import parameter and upon running the export template it will be appended it to the .qif format. File specifics and modification of exporting such information are provided in the software package you are using. More information on the export is in the example export template.

Appendices

File Formats

WebDNA uses the following file formats:

- Database format
- Shopping Cart/Order file format
- Browser info.text format
- Email format

DATABASE FORMAT

WebDNA can open up unlimited databases, each of which can have its own field structure. The number of records is only limited by the amount of RAM given to WebDNA (when used as a plug-in, you must give more RAM to WebSTAR itself). In order for WebDNA to know the name of each field in a database, the first record must contain the field names. However, since many export options do not let you save the field names as the first record, we've provided a way to separate the file with the field names from the data itself.

If WebDNA detects a file in the same location as the database, whose file extension (suffix) is .hdr instead of .db or similar, it will use the contents of the .hdr file as the list of field names.

Tab-Delimited Format: The following is an example of a tab delimited text file database. It was created in FileMaker Pro and exported using the "Tab-Delimited" format - because FileMaker does not automatically put field names across the top of tab-delimited export files, we had to manually paste the field names into the text file after FileMaker created it:

Contents of Tab-Delimited format AddressList.db file (the .db file extension is a convention only: it is not required):

The fields are separate by tabs, as usual.

Merge Format: following is an example of a database that has been exported to Merge format from a FileMaker Pro database. Notice that FileMaker automatically puts the field names across the top of the file, so you don't have to make any changes after exporting.

Contents of Merge format AddressList.db file (the .db file extension is a convention only: it is not required):

```
name,address,city,state,zip
"Bruce","11880 Central Ave","San Diego","CA","92122"
"PCS","11770 Bernardo Plaza Court","San Diego","CA","92128"
```

An example of separating the field headers from the data is the following. Suppose you export a tab-delimited list of your data, but it does not contain a field name record at the top. The database export, "data.txt", might look like the following:

```
John.....contact.....10
Grant....contact.....13
Jay.....contact.....12
```

Rather than opening the file and pasting in the field name header each time, simply create a file named "data.hdr" that contains the field names and place it in the same folder as the tab delimited database. WebDNA automatically looks for this file and does not have to be specified as an argument to the "db" parameter.

```
name.....type.....stage
```

Notice that the .hdr file should only contain one line of text and end with a carriage return. If your database is tab delimited, the .hdr file should be tab delimited. If the database is a Merge format (comma delimited) then the .hdr file should be Merge format as well.

Note: WebDNA always saves databases in tab-delimited format, regardless of what they originally were. Any Delete, Append, or Replace action will cause a database to be written to disk in tab-delimited format.

SHOPPING CART/ORDER FILE FORMAT

The shopping cart (and order file) format is a tab-delimited text file in the following format:

```
H <tab> Version <tab> Date <tab> Time <tab> Email
<tab> PayMethod <tab> AccountNum <tab> ExpMonth
<tab> ExpYear <tab> Name <tab> Company <tab> Address1
<tab> Address2 <tab> City <tab> State <tab> Zip <tab> Phone
<tab> TaxRate <tab> ShipVia <tab> ShipCost <tab> Header1
<tab> Header2 <tab> Header3 <tab> Header4 <tab> Header5
<tab> country <tab> ShipToEmail <tab> ShipToName <tab>
<tab> ShipToCompany <tab> ShipToAddress1 <tab> ShipToAddress2
<tab> ShipToCity <tab> ShipToState <tab> ShipToZip
<tab> ShipToCountry <tab> ShipToPhone <tab> Header6
<tab> Header7 <tab> Header8 <tab> Header9 <tab> Header10
<tab> Header11 <tab> Header12 <tab> Header13 <tab> Header14
<tab> Header15 <tab> Header16 <tab> Header17 <tab> Header18
<tab> Header19 <tab> Header20 <tab> Header21 <tab> Header22
<tab> Header23 <tab> Header24 <tab> Header25 <tab> Header26
<tab> Header27 <tab> Header28 <tab> Header29 <tab> Header30
<tab> Header31 <tab> Header32 <tab> Header33 <tab> Header34
<tab> Header35 <tab> Header36 <tab> Header37 <tab> Header38
<tab> Header39 <tab> Header40 <tab> NonTaxableTotal
<tab> TaxableTotal <tab> TaxTotal <tab> ShippingTotal
<tab> CartIPAddress <tab> CartUsername <tab> CartPassword
<tab> Precision <tab> TaxableShipping <tab> AuthNumber
<tab> ResponseText <tab> Status <tab> BatchNumber
<tab> ReferenceNumber <tab> SequenceNumber
<tab> ItemNumber <return>
```

Followed by multiple line items (one per sku, tab-delimited, return at end):

```
L <tab> sku <tab> quantity <tab> price <tab> taxable
<tab> canEmail <tab> unitshipCost <tab> textA
<tab> textB <tab> textC <tab> textD <tab> textE <return>
```

WebDNA saves Shopping Cart files in a folder you specify in the preferences. If the ShoppingCartFolder preference is “shopping carts/”, then WebDNA looks for a folder called “shopping carts” in the same folder as the template being displayed. If the preference is “../shopping carts/” then WebDNA looks for the folder one level “up” from the template.

Shopping Cart files stay in the shopping carts folder until they expire (24 hours of inactivity), or until the visitor presses the “Purchase” button on an invoice form. Then the file is moved to the Orders folder defined in the preferences. If you own WebMerchant, it will pick up the orders that appear

in that folder and process them accordingly. WebDNA itself will do nothing to the files once they are moved to the Orders folder.

BROWSER INFO.TXT FORMAT

The Browser Info.txt file is a list of information about various web browsers (e.g. Netscape, Microsoft Internet Explorer) that tells WebDNA what the HTML capabilities of each browser are. WebDNA can hide HTML from browsers that do not understand certain tags: for instance, Netscape 1.0 cannot understand <TABLE>, but Netscape 1.1 can. This helps you design a single web page that will automatically be tailored for each browser that visits that page.

```
Browser Name <tab> HTML Level <tab> SSL-Aware?  
Mozilla/1.0 1 T  
Mozilla/1.1 1 T  
Mozilla/1.2 2 F  
Mozilla/1. 2 T  
Mozilla/2. 2 T  
Mozilla/3. 3 T
```

The list above says that Netscape 1.0x (Mozilla/1.0) will see text inside WebDNA's [HTML1][HTML1] tags, and that it is SSL-aware.

The same is true for Netscape 1.1x (Mozilla/1.1). Microsoft's Information Explorer identifies itself as Mozilla/1.2, but MSIE does not understand SSL, so we have marked it as "F" (false) in the list above. This means WebDNA will not show any text inside [secureBrowser][secureBrowser] to MSIE browsers.

Netscape 1.3 and above are marked as [HTML2], with SSL true

Netscape 2.x and above are marked as [HTML2], with SSL true

Netscape 3.x and above are marked as [HTML3], with SSL true

Any browser not listed will be assumed to be [HTML1] and non-SSL.

As new browsers reach the market, you can modify this file so that WebDNA can display correct HTML for each one.

One thing to remember is that [HTML1], [HTML2], and [HTML3] are arbitrary classifications that do not correspond to W3.org's definitions of HTML levels. They are just 3 different classifications that you can make however you wish. If you do not include special HTML code for Netscape 3.x, then Browser

Info.txt would work better if the last line of the file said, 'Mozilla/3. 2 T' If you want to set up a site that displays information as a table to table-capable browsers and uses a non-table way to display the information to non table-capable browsers, then there is no need to use [HTML3] at all. You would only use table [HTML2] and non-table [HTML1] ways to display information. Even if you used frames, only 2 different sets of [HTML] need to be used because the frame information is put inside of a HTML comment <!> .

EMAIL FORMAT

WebDNA does not actually send the email; it writes a special file into an Email Folder, which the separate EMailer program uses to send the email. If the EMailer program is not running, no emails will be sent. The EMailer program will not erase old outgoing email files until it has successfully completed sending the email. On Macintosh systems, EMailer is a separate program, which must be running in order for emails to be sent. On Windows, an internal thread inside of WebDNA provides the email functions itself. On UNIX systems, the standard mail program built into the system itself is used; WebDNA shells out a command to the system mailer each time an email is sent.

If for some reason you wish to send emails from your own CGIs or programs (without using WebDNA's built-in [SendMail] context), you may simply write a text file into the EMailFolder specified in the EMailer preferences (Macintosh and Windows only). The file must have the following format (only the text between the lines goes into the email file). The first four standard headers are required, any others are optional. A blank line must follow the last header line. All the remaining text is part of the body of the message.

```
-----Text file saved into EMail folder-----  
to: one or more email addresses  
from: email address  
subject: subject text  
Date: full date  
Any other MIME headers  
Body text
```

Example:

```
---- /WebCatalogEngine/EMailFolder/SomeFileName.txt ---  
to: sales@webdna.us,info@webdna.us  
from: me@my domain.com  
subject: This is the subject line  
Date: Mon, 05 Jan 1998 15:59:33 -0500  
body...  
body...
```

Formulas

When your customers Add or [AddLineItem] products to their shopping cart, WebDNA obtains the price of the product in one of two ways: the price can come from a field in the database called “price”, or it can be calculated based on a formula. To prevent “hacking”, WebDNA never allows remote users to set product prices - but you can still customize pricing by creating a formula to calculate a different price based on any WebDNA tags, such as [username], [zip], or even a [math] calculation. Table 1 provides a description of each purchase-related field and the WebDNA calculation method used for each field.

For sensitive information, such as the markup on an item, the incoming web variable containing this information should never be pulled directly. Instead, add a SKU lookup to give back the retail price that also may reflect any adjustments such as a 10% discount for a repeat customer or a special.

Note: The TeaRoom example uses a formula to calculate price, taxRate, unitShipCost and overall shipping costs based on the customer’s shipping address.

Every time a product is added to the shopping cart, WebDNA calculates the item’s price and the unitShipCost as follows:

1. Look for a file called Formulas.db in the same folder as the shopping cart template itself, and use the default values for “quantity”, “price”, etc. as the initial values. If you would like to use the default price value ([price]) as a variable in the “price” calculation, the default value will be used.
2. If Formulas.db contains a “price” formula, then evaluate the WebDNA expression (in the context of the current shopping cart file, so tabs such as [zip] and [sku] are available).
3. Set the price of the product based on the calculated formula, or if no formula is found then simply use the “price” field from the product database that corresponds the item’s SKU.
4. Repeat Steps 1 through 3 above for “unitShipCost”.
5. Repeat Steps 1 through 3 above for “taxable”.
6. Repeat Steps 1 through 3 above for “taxRate”, which is applied to the entire order, not just the one item that was added. If no formula is found, then look for a form variable called “taxRate” and use that instead.

7. Repeat Steps 1 through 3 above for “shipCost”, which is applied to the entire order, not just the one item that was added. If no formula is found, then look for a form variable called “taxRate” and use that instead.

Table 1. Purchase Field Formulas and WebDNA calculation method

Field	WebDNA Calculation Method Used
Price	Lookup SKU in database, or calculate from formula (applied only to this lineItem being added).
UnitShipCost	Lookup SKU in database, or calculate from formula (applied only to this lineItem being added).
TaxRate	Optional parameter to Add command, or calculate from formula (applied to entire order file).
ShipCost	Optional parameter to Add command, or calculate from formula (applied to entire order file). This number is added to the sum total of all the unitShipCost values to arrive at the shippingTotal for the whole order.
Taxable	Lookup SKU in database, or calculate from formula (applied only to this lineItem being added). The result of the formula must be either T or F.

The GeneralStore example uses the following formula for price:

```
[lookup db=catalog.txt&lookInField=sku&value=[sku]
&returnField=price]
```

This formula looks up the price of the product in the database and returns it unchanged. This is for demonstration purposes only, because in this particular case, the formula calculates exactly the same price as though you had not specified a formula at all. To do something more complex, you might perform some calculation based on the visitor's [username] information, like so:

```
[math][showif [username]=GRANT]0.0*[/showif]
[lookup db=catalog.txt&lookInField=sku&value=[sku]
&returnField=price]
[/math]
```

This example would set the price to \$0 if the user was logged in as “GRANT”, otherwise the price would be unchanged.

Example Shipping Cost Strategies	
If your charges for shipping are...	then the formulas would be as follows:
\$6.95 + \$2.00 per additional item	<p>shipCost 4.95 (in Formulas.db)</p> <p>unitShipCost 2.00 (in Formulas.db)</p> <p>When there is 1 item in the cart, the shipTotal will be 4.95+2.00 = 6.95</p>
\$15.00 flat	<p>shipCost 15.00 (in Formulas.db)</p> <p>unitShipCost 0.00 (in Formulas.db)</p>
\$9.95 base charge + each item has its own handling charge (often based on weight)	<p>shipCost 9.95 (in Formulas.db)</p> <p>“unitShipCost” field in your product database contains numeric cost for each item. Erase unitShipCost formula from Formulas.db, so that product database field is used instead of formula. When there is 1 item in the cart, the shipTotal will be 9.95 + [unitShipCost] taken from that SKU's record in the product database.</p>
\$15.00 flat in the state of NJ, \$35.00 everywhere else	<p>shipCost [ShowIf [ShipToState]=NJ]15.00[/ShowIf][HideIf][ShipToState]=NJ]35.00[/HideIf]</p> <p>unitShipCost 0.00 (in Formulas.db) 15% of the subtotal shipCost</p>

Example Shipping Cost Strategies	
15% of the subtotal	shipCost $[\text{subTotal}] * .15$ (in Formulas.db) unitShipCost 0.00 (in Formulas.db)

ISP Sandbox

What does a WebDNA Sandbox do?

Basically it provides a way for a WebDNA admin to designate a particular folder, and its subfolders, as a WebDNA Sandbox. This means that WebDNA templates running from within that folder will not be able to view, manipulate, delete, or create any files outside of their root Sandbox folder.

This enables a WebDNA ISP to allow multiple WebDNA developers access to the same machine knowing that each sites is 'locked down' to its own root folder.

But that's not all...

Each sandbox site has its own 'WebCatalog Prefs' file called **Sandbox Prefs**, which includes nearly every preference you would find in the master WebCatalog Prefs file.

Each Sandbox has its own **Triggers** process

Each Sandbox has its own **Mailer** process

Each Sandbox has its own **Users database**

Each Sandbox has its own **Admin** section, which includes nearly all the templates you would find in the main WebDNA admin pages.

Each Sandbox has its own **Globals**, **EmailFolder**, **EmailCompleted** folder, etc....

If fact, each Sandbox has nearly every database, template, and system folder you would normally find in the main WebDNA engine folder.

Some Experimentation

Lets try a database search on an arbitrary db file that we know is outside the Sandbox...

```
[search db=../../../../test.db&eqIDdata=1]
[numfound]
[/search]
```

Results...

```
WebDNA Sandbox security does not allow access to:
*\test.db*An unknown error occurred:
DBError
../../../../test.db
```

WebDNA Changes

Here is a list of WebDNA Tags and Contexts whose behavior is altered when used inside a WebDNA Sandbox...

These contexts can only operate on files and or folders that exist within the Sandbox root folder, or the sandbox 'globals' folder when the '^' prefix is used.

```
[appendfile]
[writefile]
[renamefile]
[deletefile]
[movefile]
[copyfile]
[fileinfo]
[listfiles]
[waitforfile]
[copyfolder]
[createfolder]
[deletefolder]
```

These operations can only operate on a cart or files that exists within the sandbox root, or the sandbox 'globals' folder when the '^' prefix is used.

```
[orderfile]
[purchase]
[removelineitem]
[clearlineitem]
[setlineitems]
[setheader]
```

Will only flush the databases that exists within the sandbox root or globals folder.

```
[flushdatabases]
```

These contexts will only operate on database files that exist within the sandbox root or globals folder.

```
[commitdatabase]
[listdatabases]
[closedatabase]
[listfields]
[search]
[lookup]
[replace]
[append]
[delete]
```

Uses the 'local' sandbox 'users.db' file.

```
[protect]
```

Both these tags use the 'format' preferences in the local sandbox prefs file.

```
[date]
[time]
```

Can only include a file that exists within the sandbox root or globals folder.

```
[include]
```

Uses the local sandbox email settings. Resulting email files are written to the sandbox 'EmailFolder'.

```
[sendmail]
```

These contexts require special handling, discussed later.

```
[object]
[dos]
[applescript]
```

Uses the local sandbox 'StandardConversions.db' file. If a database is specified, it must exist within the sandbox root or globals folder.

```
[convertchars]
```

Can only be used with a database that exists within the sandbox root or globals folder.

```
[convertwords]
```

Misc. Sandbox Notes

The '/' path prefix will be relative to the Sandbox root folder.

The '^' global prefix will refer to the Sandbox globals folder and not the main WebDNA globals folder.

The 'absolute path' specifier '*' can still be used with a sandbox, as long as it refers to a path that exists within the sandbox root or globals folder.

WebDNA error messages will be retrieved from the sandbox ErrorMessage.db, and not the main WebDNA ErrorMessage.db

WebDNA error logs and debug files will be created within the sandbox system folder, and not the main WebCatalogEngine folder.

Shell, DOS, Applescript

The **[shell]**, **[dos]**, and **[applescript]** contexts require special handling when used within a WebDNA Sandbox.

Shell, DOS, and Applescript code can no longer be explicitly defined within these contexts, when used in a WebDNA Sandbox. Instead, the desired code must be submitted to the WebDNA admin for approval. If approved, the 'code snippet' is added to the WebDNA Sandbox 'Scripts' database, with a unique ID code. This ID code is then used by the Sandbox programmer.

For example...

A WebDNA sandbox programmer wants to execute the DOS command. "**dir c:**". The programmer submits the code snippet, "**dir c:**", to the WebDNA admin. The admin approves the code, and inserts the code into the WebDNA Sandbox Scripts database with a unique ID of '111'. The WebDNA admin then returns the ID to the Sandbox programmer. The Sandbox programmer then uses the ID as follows...

```
[DOS scriptID=111] [/DOS]
```

The same applies when using the **[shell]** context.

The **[object]** context is handled in a slightly different way. For this case, the submitted 'code snippet' is the 'parameter list' that would normally be passed into the context.

For example...

```
[OBJECT objname=SystemInfo.SysInfo.1&type=0[!]  
[!]&call=GetFreeDiskSpace[!]  
[!]&param1=C:\&param1type=str] [/OBJECT]
```

would become...

```
[OBJECT SCRIPTID=123] [/OBJECT]
```

The WebDNA admin having recorded the text:

```
"objname=SystemInfo.SysInfo.1&type=0[!]  
[!]&call=GetFreeDiskSpace[!]  
[!]&param1=C:\&param1type=str"
```

into the Sandbox scripts database with an ID of 123.

Pre and Post Parse Scripts

The pre-parse and post-parse scripts exist as two 'special' WebDNA files in the WebCatalog engine folder, or WebDNA Sandbox system folder. These are named, as you might guess, 'PreParseScript' and 'PostParseScript'. When enabled, any WebDNA that is contained in those scripts is processed before, or after, every WebDNA page request.

The Pre-Parse Script

The pre-parse script is an ideal place to put 'global' function or variable definitions. It is also a great place to put HTML comment blocks that should appear at the top of WebDNA page results. When combined with a WebDNA Sandbox, this level of functionality can be considered as 'application' level functionality, as each WebDNA Sandbox will use a separate set of pre-parse and post-parse scripts.

Here is an example of a pre-parse script you could use to initialize a series of function definitions. In this example the function definitions exist as separate include files in a sub-folder called, 'FunctionDefs'.

Contents of the 'PreParseScript' file...

```
[!] An example Pre-Parse Script that loads function  
definitions [!]  
[!]  
Include My Function Definitions.
```

```

[!/]

[text]results=
[listfiles path=^FunctionDefs]
[showif [isfile]=T]
[include ^FunctionDefs/[filename]]
[/showif]
[/listfiles]
[/text]

```

If this example pre-parse script was 'live', then every WebDNA page request would have access to any function definitions that were processed as a result of the script execution.

The Post-Parse Script

The post-parse script is executed at the end of every WebDNA page request. This enables the WebDNA programmer to 'wrap' WebDNA code around the results of every WebDNA template that is processed. So this is an ideal place to insert code that generates 'global' (or 'application' when inside WebDNA Sandbox) header or footer messages. Or as a place to put server, or Sandbox, 'logging' code. It can even be used to insert WebDNA code that will 'strip' or modify the page results, perhaps to remove unwanted white-space from the HTML before it is return to the client.

There is a new, very important, WebDNA tag that you will almost always use inside the post-parse script. This tag is:

[WEBDNA_TEMPLATE_RESULTS]. This tag represents the results of the requested WebDNA template. This tag allows you to 'wrap' you own WebDNA code around the page results. So, if you enable the post-parse script feature, you will need to include the [WEBDNA_TEMPLATE_RESULTS] tag, somewhere in the WebDNA code.

Here is an example of a post-parse script you could use to insert an HTML footer message at the bottom of every WebDNA page request.

Contents of the 'PostParseScript' file...

```

[!] Example code to place a 'footer' message at the bottom
of every page request [!/]

[showif [WEBDNA_TEMPLATE_RESULTS]^

```

How to use the scripts

Both the Pre-Parse and Post-Parse scripts are disabled by default. Even if the scripts are 'enabled', the stock scripts contain little to no 'active' WebDNA code. The shipping version of WebDNA 5.0 may include pre-defined functions that will be loaded by the pre-parse script.

You enable the scripts via the WebDNA admin, preferences page. The scripts can be independently enabled or disabled from that page. There is also a 'link', for each script, that will lead to a page where the script can be edited and saved.

IMPORTANT:

You should have ftp access to the folder containing the scripts, before trying to enable and edit them. This is in case you insert bad HTML or WebDNA code that renders all pages unreadable (which would render the admin pages unreadable as well). Ftp access to the script files will ensure that you have a way to 'correct' fatal errors in the script files. For WebDNA Sandbox sites, malformed scripts would only effect pages served from that Sandbox.

To enable, disable, view, or edit the pre_parse or post-parse scripts, select the 'Prefs/...Admin...' menu item (in the lab 'source' view). When the admin page is displayed, click on the 'Preferences' link in the left pane. You will see the pre-parse and post-parse script options at the bottom of the 'General' section.

Triggers

Triggers provide a mechanism for doing something on a regular timed basis, or when a certain action occurs. Currently only time-based triggers are provided, but in the future new types of triggers we will added to perform an action when ever a database is modified or a template displayed.

Triggers do their work by simulating a browser hit to a URL. They act as if you had manually used a browser to reload a page at a particular time each day. This gives you the flexibility to create as much complex WebDNA as you like, and to test it by simply using a browser to visit that URL. Once you have finished creating and testing the template, then enter its URL into a trigger and it will be executed on schedule from then on.

The URL must be of the form <http://www.server.com/folder/file.ext> (the same as you would see in a browser window—in fact, it is probably easiest to

simply copy the URL directly from your browser (window). There is no restriction on the web site in the URL, so you can actually have triggers that hit any web server in the world.

Because it is possible that the URL will fail for some reason (timeout, bad connection, bad password), triggers have a timeout and retry interval. These numbers are used to tell the trigger how long it should wait before attempting the URL again. Triggers look for a string of text (which you specify) that tell them the trigger was successfully executed. Often the text "<html>" is sufficient, but you can put more sophisticated WebDNA into a page to create more detailed success information.

The single Triggers.db file must be in the main program folder (where Users.db, ErrorMessage.db, etc. reside). An example of a useful time-based trigger is one that looks through the ShoppingCarts folder once per hour and deletes any that are more than 24 hours old. Another example is a trigger that looks for new order files in the Orders folder and initiates a credit card transaction using credit card software such as ICVerify, MacAuthorize, or CyberCash. Once the transaction is cleared, the trigger URL's WebDNA could continue by sending fulfillment emails and updating an inventory database.

Trigger Fields:

SKU: A unique number that makes it easier to identify a particular trigger

Trigger: Currently only TIME is allowed here, but in the future we will add values like APPEND, DELETE, REPLACE, SHOWPAGE, etc.

Param: For TIME triggers, this is a specially formatted string of numbers and asterisks that represent the time the trigger should execute. For example, to cause a trigger to execute 5 seconds after each minute, the text would be Y M D H M S (Year Month Day Hour Minute Second) "* * * * * 5". To cause a trigger to execute at 9:15 PM every day, the text would be "* * * 21 15 0".

Param Field Values:

Year	* or actual year such as 1998
Month	* or month of year such as 6 for June
Day	* or day of month such as 28
Hour	* or hour of day (24 hour clock) such as 23 (11 PM)

Minute	* or minute of hour such as 59
Second	* or second of minute such as 15
NextExecute	This value gets changed automatically each time a trigger is executed. It is changed to the date and time of the next scheduled execution of this trigger. If you do not want a trigger to start until a future date, you can preset this to the first date you want it to execute. After that, it is updated automatically.
Enable	T or F to enable or disable the trigger
ExecuteURL	Full URL to the template that you want executed at trigger time
User	Optional username for this page. This is the same as an authenticated username that the [protect] tag uses. Requiring a username/password enables you to create triggers that outside visitors cannot view.
Pass	Optional password for this page. This is the same as an authenticated password that the [protect] tag uses. Requiring a username/password enables you to create triggers that outside visitors cannot view.
WasGood	A string of text that is returned from the URL that indicates the trigger was successfully executed. The trigger looks for this text anywhere in the returned HTML from the URL page. Often <html> is sufficient to tell the trigger that it successfully received the page.
TimeoutSeconds	Number of seconds to wait for the URL to complete before giving up and trying again
RetrySeconds	Number of seconds to wait before retrying the trigger.

License and Limited Warranty Agreement

WebDNA Software Corp. grants you a personal, non-transferable, non-exclusive license to use this copy of the software program and the accompanying documentation ("Software") according to the following terms and conditions:

1. License and Restrictions

(a) This agreement grants you the right ("License") of use of one copy of the enclosed WebDNA Software Corp software on any single computer at any time. You may not copy or load the Software onto the same or additional computers or use the Software on a network unless you have purchased additional licenses for each instance of use.

(b) The Software is owned by WebDNA Software Corp and is protected by U.S. copyright laws and international treaty provisions. You must treat the Software just like a book or other copyrighted item, except that you may (i) make one copy of the Software program for backup or archival purposes, and (ii) transfer the Software program onto a single computer's hard disk or solid state storage. You may not copy any Software documentation provided to you.

(c) This Agreement is your proof of License to exercise the rights granted herein, and must be retained by you. You may not lease or rent the Software but you may transfer your rights under this Agreement on a permanent basis, provided that you transfer the Software, accompanying documentation, as well as this Agreement, and that the recipient agrees to the terms of this Agreement. You may not modify, merge, decompile or reverse-engineer the Software, and you may not remove or obscure the WebDNA Software Corp, trademark or copyright notices in the Software or documentation.

(d) If you are an agency, department or other entity of the United States government, you shall be subject to restrictions of Restricted Rights for computer software developed at private expense as set forth in FAR 52.227-14, FAR 52.227-19 and DOD FAR Supplement 252227-7013(c)(1)(ii), including Alternate III and successors thereof, as applicable.

(e) Periodically, the software will make a secure SSL connection to WSC's server and report the following information: Product Name, Product Version, Serial Number, Host Name. The server will log the information and may send back a plain text message that will be displayed on the client's Administration page. This message will likely contain information about available product updates and patches.

2. Title

You acknowledge and agree that all right, title and interest in and to the Software, and all intellectual property rights therein, shall remain the property of WebDNA Software Corp. You have a license to use the Software only and no ownership or proprietary rights to the Software are transferred to you.

3. Software Limited Liability and Disclaimer

WEBDNA SOFTWARE CORP. MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS IN CONNECTION WITH THE SOFTWARE, EXPRESS, IMPLIED, STATUTORY OR IN ANY OTHER PROVISION OF THE LICENSE OR COMMUNICATION BETWEEN YOU AND WEBDNA SOFTWARE CORP. WEBDNA SOFTWARE CORP. SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU.

4. Limitation of Liability

WEBDNA SOFTWARE CORP.'S LIABILITY UNDER OR ARISING OUT OF THIS LICENSE WITH RESPECT TO THE SOFTWARE, WHETHER FOR BREACH OF CONTRACT OR TORT OR OTHERWISE, SHALL NOT EXCEED THE AMOUNT THAT YOU PAID FOR THE SOFTWARE. IN NO EVENT WILL WEBDNA SOFTWARE CORP. BE LIABLE FOR ANY DAMAGES FOR LOSS OF DATA, LOST PROFITS, COST OF COVER OR OTHER SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES ARISING FROM THE USE OF THE SOFTWARE OR ACCOMPANYING DOCUMENTATION, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY. THIS LIMITATION WILL APPLY EVEN IF WEBDNA SOFTWARE CORP OR AN AUTHORIZED DISTRIBUTOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE, AND NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY, SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

Support

Though WebDNA Software Corporation is not responsible for maintaining or helping you use the Software, WebDNA Software Corporation does, at its discretion, offer support via the following options:

- Our Web Site: <http://www.webdna.us>
- Our support forum and mailing lists:
Sign up at <http://www.webdna.us>
- Direct email support: support@webdna.us

Glossary

Command – WebDNA commands direct WebDNA to perform various functions. They are sent directly to the application via URL's. WebDNA commands are embedded in a URL and give an explicit context to a particular template being displayed.

Context – A WebDNA context encloses a block of text and requires a beginning and ending tag. Like HTML enclosing tags, the ending WebDNA context tag is specified with the name of the context preceded by a forward slash “/”. Like WebDNA tags, contexts are enclosed in square brackets rather than angle brackets as well.

HTML Form – HTML forms are the standard means for having a visitor send information to a web server. All the data contained in a form, both hidden data and data entered by the visitor, is bundled up and sent to WebDNA via the web server. An HTML form is simply a collection of <input> fields with an associated name/value pair. The form's data is sent to the server in a manner similar to the name/value pairs used to specify WebDNA parameters.

Internet Storefront – See *web store*.

Store – See *web store*.

StoreBuilder – StoreBuilder is an open set of WebDNA templates that allow you to create as many eCommerce sites as you web server will allow.

Storefront – See *web store*.

Tag – A WebDNA tag is just like an HTML tag with one major exception: It never “exists” as far as a browser is concerned. Instead, it is replaced by text (any valid HTML) on the server by WebDNA before being sent to the browser. Think of WebDNA tags as server tags, and HTML tags as browser (client-side) tags.

Template – A predefined web page that can control page layout (i.e., text fonts, colors, page design, colors and graphics), as well as functionality (e.g., emailer form, product order form, customer login page, etc.).

Trigger – *In WebDNA, a trigger provides a mechanism for doing something on a regular timed basis, or when a certain action occurs.*

Web – (a.k.a., the World Wide Web or WWW) an Internet feature housing web sites and their corresponding web pages from Internet sites all over the world.

WebDNA (the product line) – WebDNA products are Internet applications that work with your existing web server allowing you to design web applications with all the features found in the leading sites.

WebDNA (the language) – WebDNA is a scripting language for creating Web sites. It adds functionality to a web server, and is used to tell the WebDNA products what to do. WebDNA exists in HTML files on your server or within URL's sent by your browser.

WebMerchant – WebMerchant is a sophisticated program, effortlessly handling the automated payment processing capabilities of CyberCash, ICVerify, MacAuthorize, and other payment systems.

Web Store – An eCommerce catalog used to advertise products and services through the World Wide Web portion of the Internet (a.k.a., store, storefront, and other names).

Index

- About This Manual
 - Structure, i
 - Advanced WebMerchant Topics, 266
 - Archive
 - Talk List, 251
 - Banner Ads
 - Generating Online, 252
 - Browser Info
 - GetMIMEHeader Tag, 190
 - Brower Info
 - GetCookie Tag, 189
 - Browser Info, 189
 - BrowserName Tag, 189
 - IPAddress Tag, 190
 - IsSecureClient Tag, 190
 - ListCookies Context, 190
 - ListMIMEHeaders Context, 192
 - Referrer Tag, 193
 - SetCookie Tag, 193
 - SetMIMEHeader Tag, 194
 - Browser Info.txt Format, 271
 - Commands
 - Add, 98
 - Append, 77
 - Clear, 104
 - Command=Notation, 247
 - Delete, 78
 - FlushCache, 178
 - FlushDatabases, 180
 - NewCart, 107
 - NewCartSearch, 106
 - Purchase, 110
 - Quit, 89
 - Raw, 159
 - Remove, 117
 - Replace, 93
 - Replacing Commands with Contexts, 23
 - Search, 65
 - Searching, 21
 - Show Cart, 122
 - ShowPage, 129
 - Technical
 - Raw Command, 184
 - Using Contexts vs. Commands, 21
 - Using WebDNA, 246
 - Context
 - LineItems, 105
 - Contexts
 - ! Comment, 241
 - AddLineItem, 101
 - Append, 73
 - AppendFile, 79, 165
 - AppleScript, 174
 - BoldWords, 140
 - Capitalize, 141
 - Context Parameters, 245
 - Context Variables, 245
 - ConvertChars, 142
 - ConvertWords**, 144
 - CountChars, 143
 - CountWords, 143
 - DDESend, 176
 - Decrypt, 146
 - DOS, 177
 - Encrypt, 146
 - ExclusiveLock, 75, 81
 - FileInfo, 170
 - Format, 132, 148
 - FormVariables, 220
 - FoundItems Context, 63
 - Grep**, 152
 - HideIf, 124
 - HTML1, 125
 - HTML2. *See* HTML1 Context
 - HTML3. *See* HTML1 Context
 - IfThenElse**, 126
 - Input, 153
 - Interpret, 181
 - ListCookies, 190
 - ListDatabases, 82
 - ListFields, 83
 - ListFiles, 171
 - ListMIMEHeaders, 192
 - ListPath, 154
 - ListVariables**, 84, 194, 199, 201, 204, 206, 217, 218, 222, 224, 228, 231
 - ListWords, 156
 - Loop, 226
 - LowerCase**, 157
 - Math, 134
 - Middle, 158
 - Object, 182
-

- OrderFile, 107
- Raw, 158
- RemoveHTML**, 160
- Replace, 90
- ReplaceFoundItems**, 94
- Replacing Commands with Contexts, 23
- ReturnRaw, 185
- Search Context, 64
- Searching, 21
- SendMail, 238
- SetHeader, 118
- SetLineItem, 120
- Shell, 177
- ShowIf, 128
- ShowNext, 69
- Spawn, 185
- SQL, 71, 96
- Switch Case, 128
- Tag Parameters, 245
- TCPConnect, 186
- TCPSend, 187
- Text, 160
- UnURL, 162
- UpperCase, 163
- URL, 163
- Using Contexts vs. Commands, 21
- Using WebDNA, 244
- WaitForFile, 173
- WriteFile, 173
- Database Design, 13
- Database Format, 268
- Databases, 73
 - Append Command, 77
 - Append Context, 73
 - AppendFile Context, 79
 - CloseDatabase Tag, 73
 - CommitDatabase Tag, 73
 - Delete Command, 78
 - Delete Tag, 81
 - ExclusiveLock Context, 75, 81
 - FlushDatabases Tag, 82
 - ListDatabases Context, 82
 - ListFields Context, 83
 - LookUp Tag, 83
 - Quit Command, 89
 - Replace Commands, 93
 - Replace Context, 90
 - ReplaceFoundItems Context**, 94
 - SQL Context, 96
- Dates and Times, 131
 - Date Tag, 131
 - Format Context, 132
 - Math Context, 134
 - Time Tag, 140
- Dreamweaver Integration, 253
- Email Format, 272
- Encrypting Templates, 249
 - Header
 - Encrypting, 250
- File Formats, 268
- Files
 - Uploading, 262, 263
- Files and Folders, 165
 - AppendFile Context, 165
 - CopyFile Tag, 167
 - CopyFolder Tag**, 166, 167, 168
 - CreateFolder Tag, 167
 - DeleteFile Tag, 167
 - DeleteFolder Tag, 168
 - FileInfo Context, 170
 - ListFiles Context, 171
 - MoveFile Tag, 172
 - RenameFile Tag**, 172
 - WaitForFile Context, 173
 - WriteFile Context, 173
- Form Variables. *See* WebDNA Tags
- Formats
 - Browser info.txt, 271
 - Database, 268
 - Email, 272
 - File, 268
 - Order File, 270
 - Shopping Cart, 270
- Formulas, 273
- Generating Online Banner Ads, 252
- Glossary, 289
- HTML Forms, 11
- License, 286
- ListCookies Context
 - Setting Cookies, 191
- Logging Information, 24
- Math Context
 - Dates, 136
 - Functions, 138
 - Times, 137

- Variables, 138
- Miscellaneous**, 217
 - ! Comment Context, 241
 - FormVariables Context, 220
 - FreeMemory Tag, 221
 - Include Tag, 223
 - LastRandom Tag, 224
 - ListVariables Context**, 84, 194, 199, 201, 204, 206, 217, 218, 222, 224, 228, 231
 - Loop Context, 226
 - Platform Tag, 227
 - Random Tag, 227
 - SendMail Context, 238
 - ThisURL Tag, 241
 - Version Tag, 241
- Near Real Time Processing, 15
- Order File Format, 270
- Order Files
 - Using External Accounting Software with, 267
- Password
 - Protect Tag, 165
 - Username Tag, 165
- Passwords, 164, 194
 - Authenticate Tag, 164
- POP Mailbox
 - Shared
 - Using, 251
- Raw Command, 159
- Real Time Processing, 15
- Searching
 - FoundItems Context, 63
 - LookUpTag, 63
 - Search Command, 65
 - Search Context, 64
 - ShowNext Context, 69
 - SQL Context, 71
- Security, 255
 - MacIntosh WebDNA Security Notes, 256, 263, 264
 - Threats
 - Areas to Watch for, 262
- Security Threats, 262
- SendMail Context
 - Mailer Error Codes, 239
 - Header Fields, 240
- Shared POP Mailbox
 - Using, 251
- Shopping
 - Add Command, 98
 - AddLineItem Context, 101
 - Cart Tag, 104
 - Clear Command, 104
 - ClearLineItems Tag, 105
 - LineItems Context, 105
 - NewCart Command, 107
 - NewCartSearch Command, 106
 - OrderFile Context, 107
 - Purchase Command, 110
 - Purchase Targe, 110
 - Remove Command, 117
 - RemoveLineItem Tag, 117
 - SetHeader Context, 118
 - SetLineItem Context, 120
 - Show Cart Command, 122
 - ValidCard Tag, 123
- Shopping Cart Format, 270
- Shopping Cart Transaction Processing, 15
 - Order Collection, 17
 - Order Management, 19
 - Order Processing, 18
- Showing and Hiding, 124
 - HideIf Context, 124
 - HTML1 Context, 125
 - HTML2 Context. *See* HTML1 Context
 - HTML3 Context. *See* HTML1 Context
 - IFThenElse Context**, 126
 - ShowIf Context, 128
 - ShowPage Command, 129
 - Switch Case Context, 128
- Site Design, 12
- Subscription
 - Talk List, 251
- Support, 288
- Tags
 - Authenticate, 164
 - BrowserName, 189
 - Cart, 104
 - ClearLineItems, 105
 - CloseDatabase, 73
 - Command, 175
 - CommitDatabase, 73
 - CopyFile, 167
 - CopyFolder**, 166, 167, 168
 - CreateFolder, 167

- Date, 131
- DDEConnect, 175
- Delete, 81
- DeleteFile, 167
- DeleteFolder, 168
- ElapsedTime, 178
- FlushDatabase, 179
- FlushDatabases, 82
- FreeMemory, 221
- GetCookie, 189
- GetMIMEHeader, 190
- Header
 - Encrypting, 250
- Include, 223
- IPAddress, 190
- IsSecureClient, 190
- LastRandom, 224
- LookUp, 83
- LookUpTag, 63
- MoveFile, 172
- Password, 165
- Platform, 227
- Protect, 165
- Purchase, 110
- Random, 227
- Redirect, 184
- Referrer, 193
- RemoveLineItem, 117
- RenameFile**, 172
- SetCookie, 193
- SetMIMEHeader, 194
- ThisURL, 241
- Time, 140
- Username, 165
- ValidCard, 123
- Version, 189, 241
- WebDNA, Using, 242
- Talk List
 - Subscription and Archive, 251
- TeaRoom Database
 - Acknowledging the Order, 59
 - Adding Items to the Shopping Cart, 38
 - Development of, 30, 60
 - Entering the Site, 31
 - Shopping for Products By Category, 34
 - The TeaRoom Database, 31
 - Tutorial
 - Overview, 30
 - Using the Product Detail Page, 47
 - Using the Purchase/Invoice Page, 51
 - Using the Shopping Cart Page, 43
- Technical, 174
 - AppleScript Context, 174
 - Command Tag, 175
 - DDEConnect Context, 175
 - DDESend Context, 176
 - DOS Context, 177
 - ElapsedTime Tag, 178
 - FlushCache Command, 178
 - FlushDatabase Tag, 179
 - FlushDatabases Command, 180
 - Interpret Context, 181
 - Object Context, 182
 - Raw Command, 184
 - Redirect Tag, 184
 - ReturnRaw Context, 185
 - Shell Context, 177
 - Spawn Context, 185
 - TCPConnect Context, 186
 - TCPSEND Context, 187
 - Version Tag, 189
- Template Design, 14
- Templates
 - Encrypting, 249
- Text Manipulation, 140
 - BoldWords Context, 140
 - Capitalize Context, 141
 - ConvertChars Context, 142
 - ConvertWords Context**, 144
 - CountChars Context, 143
 - CountWords Context, 143
 - Decrypt Context, 146
 - Encrypt Context, 146
 - Format Context, 148
 - Grep Context**, 152
 - Input Context, 153
 - ListPath Context, 154
 - ListWords Context, 156
 - LowerCase Context**, 157
 - Middle Context, 158
 - Raw Command, 159
 - Raw Context, 158
 - RemoveHTML Context**, 160
 - Text Context, 160
 - UnURL Context, 162

- UpperCase Context, 163
- URL Context, 163
- Topics
 - Advanced WebMerchant, 266
- Triggers, 3, 277, 281, 283
- Understanding WebDNA, 1
- Uploading Files, 262, 263
- Using a Text Editor vs. HTML Editor, 19
- Using Shared POP Mailbox, 251
- Using WebDNA Commands, 246
- Using WebDNA Contexts, 244
 - Context Parameters, 245
 - Context Variables, 245
 - Tag Parameters, 245
- Warranty Agreement, 286
- Web Server
 - How WebDNA Acts on a Web Server Request, 27
- WebDNA
 - Advanced Uses of, 249
 - How WebDNA Acts on a Web Server Request, 27
 - Preparing Your Site for Use With, 28
 - Request Processing With, 29
 - Theory of Operation, 26
 - Tutorial, 26
 - What is WebDNA, 26
- WebDNA Advanced Uses
 - Dreamweaver Integration, 253
 - Encrypting Templates, 249
 - Generating Online Banner Ads, 252
 - Security, 255
 - Talk List Subscription and Archive, 251
 - Using Shared POP Mailbox, 251
- WebDNA Benefits, 2
- WebDNA Commands, 7
- WebDNA Contexts, 4
- WebDNA Parameters, 8
- WebDNA Reference, 62
- WebDNA Tapes, 3
- WebDNA Tags
 - Form Variables, 244
 - Italic Text, 243
 - Parameters, 242
 - Paths, 243
 - Preferences, 242
 - Using, 242
- WebMerchant
 - Advanced Topics, 266
- WebMerchant Advanced Topics
 - Account Authorizer using, 267
 - External Accounting Software, using with order files, 267
- What is WebDNA, 1
- Working with Tables, 20