Zope
Document Template
Markup Language
Reference

Document Revision 2.1.0, 1999/12/17
For Zope version 2.1.0

Modified by Pamela Crosby

Table of Contents

List of Tables vii

Introduction 1

General Information 3

- DTML Tag Syntax 3
- Server Side Include Format Syntax 5
- Alternate Python String Format Syntax 5
- Common Tag Attributes 7
  - The name attribute 7
  - The expr attribute 7
    - Expression syntax 7
    - Variable lookup 8
    - The special namespace variable, _ 8
- Name Lookup 15
- Access Control 18
- Using Document Templates from Python 21
  - Creating document templates 21
  - Using document templates 22
  - Using document templates with ZPublisher 22

The dtml-var Tag 23

- Custom, Special, C, and Empty Formats 23
- Null Values 24
- Truncation 25
- A dtml-*var* Tag Example, the Default Document Source 25

Conditional Insertion, the dtml-if and dtml-unless Tags 27

Iterative Insertion, the dtml-in Tag 29

- The dtml-else Tag as an Intermediate Tag in the dtml-in Tag 29
- Variables Defined by the dtml-in Tag 31
- Summary Statistics 33
- Grouping Variables 33
- Batch Processing 33
  - Orphan rows 35
  - Overlapping batches 37
  - Showing row number and row data in previous and next batch hyperlinks. 37
  - Showing information about multiple batches 39

Displaying Objects with the dtml-with Tag 41

- Using the only Attribute to Limit the Namespace 41

Multiple assignments with the dtml-let Tag 42

Evaluating Names or Expressions without Generating Text Using the dtml-call Tag . 43
Reporting Errors with the dtml-raise Tag 45
Exception Handling with the dtml-try Tag 46
The dtml-try tag optional dtml-else block 47
The dtml-try tag optional dtml-finally block 47
Excluding Source Text with the dtml-comment Tag 48
Returning Data using the dtml-return tag 49
Displaying Information Hierarchically: the dtml-tree tag 50
Sending Mail: the dtml-sendmail tag 54
Sending Attachments with the dtml-mime Tag 56
Appendix A, Date-time data 57

List of Tables

TABLE 1.	Expression examples	8
TABLE 2.	Available attributes in the special variable, _	9
TABLE 3.	Attributes defined by the math module	10
TABLE 4.	Attributes defined by the random module	11
TABLE 5.	Attributes defined by the string module	12
TABLE 6.	Attributes defined by the whrandom module	13
TABLE 7.	Simplest-case steps for looking up names	15
TABLE 8.	Zope-defined Web request variables	16
TABLE 9.	Attributes of the REQUEST variable.	16
TABLE 10.	Attributes of the RESPONSE variable	17
TABLE 11.	CGI-defined Web request variables	17
TABLE 12.	Document template classes	21
TABLE 13.	Standard document template creation arguments.	21
TABLE 14.	Standard arguments for calling document templates.	22
TABLE 15.	dtml-var tag attributes	23
TABLE 16.	Special formats that may be used with the *var* tag *fmt* attribute	24
TABLE 17.	C-style specifiers for the *fmt* attribute	24
TABLE 18.	*dtml-in* tag attributes	30
TABLE 19.	Item variables defined by the dtml-in tag	32
TABLE 20.	Summary statistic variables defined by the dtml-*in* tag	33
TABLE 21.	Special variables for group processing	33
TABLE 22.	Batch-processing variables	34
TABLE 23.	Attributes of batch objects used when iterating over *next-batches* and *previous-batches* variables.	34
TABLE 24.		35
TABLE 24.	Query strings and previous batch URL and next batch URL for the batches shown in figure 7	37
TABLE 25.	dtml-tree tag attributes.	51
TABLE 26.	Variables set by the dtml-tree tag when rendering sub-objects.	52
TABLE 27.	Variable that influence the dtml-tree tag	53
TABLE 28.	dtml-sendmail tag attributes	55
TABLE 29.	*dtml-mime* tag attributes	56
TABLE 30.	Custom formats for date-time data	57

Introduction

Audience: Developer

The Zope Document Template Markup Language (DTML) is a facility for generating textual information using a template document and application information stored in Zope. It is used in Zope primarily to generate Hypertext Markup Language (HTML) files, but it can also be used to create other types of textual information. For example, it is used to generate Structured Query Language (SQL) commands in Zope SQL Methods.

The DTML facility is used to convert from document template source text to rendered text. Document template source text consists of ordinary text interspersed with DTML “markup” tags.

Purpose of DTML Reference

The purpose of the DTML Reference is to provide a reference dictionary document for the many DTML tags available in Zope and to help programmers in understanding how to implement DTML. The first portion of the guide describes the syntax of DTML tags and provides a number of simple examples of document template source texts. The second section describes the DTML tags in detail.

General Information

DTML Tag Syntax

The DTML Tag is supported by three syntaxes.[1] This includes the DTML document templates, server-side includes and Python string formats. When using document templates from Python, the syntax used depends on the document template class used (or subclasses). The server-side-include new DTML syntax described here is used by the classes DocumentTemplate.HTML and DocumentTemplate.HTMLFile.

The syntax formats are shown below:

- `<dtml-name>`
- `%(name)x`
- `<!--#name-->`

The syntax used by DTML to indicate text substitution is based standard tag name attribute format used in similar templates. A DTML tag is of the form:

```
<dtml-tag name attribute1="value1" attribute2="value2" ... >
```

The tag name identifies the tag type. Following the tag name, are typically one or more attributes which indicate where the tag's data is found and how that data is to be inserted. Sometimes, attribute values can be omitted and quotation marks around attribute values can be omitted if the values do not contain a space, tab, new-line character, equal sign or double quotation marks.

```
<dtml-var date fmt=Date>
<dtml-var name="standard_html_header">
<dtml-with subfolder>
```

The most common tag is the *dtml-var* tag. The *var* tag is used to substitute variable data into text. Suppose we want to create a greeting with the variable, `input_name`. We might use a document template like the following:

```
Hello <dtml-var name="input_name" capitalize>!
```

This example uses two attributes, *name* and *capitalize*. The *name* attribute is used to specify a variable name. Typically, a variable name refers to data in World Wide Web (WWW) requests or properties of Zope objects, like folders. Because of the *name* attribute's frequent usage, there exists a hand version of the *name* attribute in which the attribute name is omitted:

```
Hello <dtml-var input_name capitalize>!
```

When using the shorthand form of the *name* attribute, the value is the first attribute in the tag and is not enclosed in quotation marks.

1. It is also possible to define additional syntaxes, although the mechanism for doing this is not currently documented. For example, a syntax that is similar to the syntax used by active server pages has been developed.

A similar shorthand notation exists for the *dtml-expr* attribute. The *expr* attribute is used to provide computational expressions, as in:

```
<dtml-if expr="age > 18">
```

This may be shortened by eliminating the attribute name, as in:

```
<dtml-if "age > 18">
```

Like in the *name* attribute, the attribute value is the first attribute in the tag and is not enclosed in quotation marks.

The *capitalize* attribute illustrates the use of an attribute in which a value, or argument, is not defined[1]. The *capitalize* attribute indicates that the first letter of the inserted text should be capitalized. Suppose the document template source from the previous example is evaluated with the input name, “`world`”, then the text output would be:

```
Hello World!
```

The *dtml-var* tag is called a singleton tag, because it does not contain any other tags. Tags which are duplex contain bracketed text which may contain other DTML tags. Duplex tags require a matching end tag. The name of the end tag is the same as the start tag, except that it contains a "`/`" or an "`end`" prefix. End tags do not have attributes. A commonly used duplex tag is the *dtml-if* tag:

```
<dtml-if input_name>
   Hello <dtml-var input_name>.
</dtml-if>
```

In this example, if the variable, `input_name`, has not been provided or is an empty string, then the greeting is omitted.

A non-empty tag can also have intermediate tags. These intermediate tags serve to break the non-empty tag into two or more sections. For example the *dtml-if* tag can use an intermediate *dtml-else* tag, as in:

```
<dtml-if input_name>
   Hello <dtml-var input_name>.
<dtml-else>
   Have we been introduced?
<dtml-endif>
```

Note that in this case, the alternate prefix of the end tag is used.

1. Actually, all attributes have values. Certain attributes, like the *capitalize* attribute, have default values which are set when a value is not provided within the tag. The only case in which a value must be provided for this type of attribute is when the attribute is the first attribute in the tag. Without a value, the attribute would be confused with a *name* attribute value. For attributes like capitalize, a value of "yes", "on" or 1 is usually provided, as in: `<dtml-var capitalize=1 name=id>`. The *capitalize* attribute is an example of a flag. The presence of a flag typically indicates that some normally disabled option should be enabled, thus the values of "yes", "on" or 1.

Intermediate tags can have attributes, as in:

```
<dtml-if input_name>
   Hello <dtml-var input_name>.
<dtml-elif nick_name>
   Hi <dtml-var nick_name>.
<dtml-else>
   Have we been introduced?
</dtml-if>
```

In the example above, there is one non-empty tag, *dtml-if* that uses two intermediate tags, *dtml-elif* and *dtml-else*, and an end tag, */dtml-if*.

Server Side Include Format Syntax

Any number of line endings, tabs or spaces may be placed between the pound character (#), the tag name, attributes or the end of a tag. This format is still valid for the older versions of Zope and DTML. The server side includes format is **<!--#command tag1="value1" tag2="value2"-->**. The older versions of Zope used this format for the dtml code. You will still see examples using this syntax. For example, the following are all valid tags which are based on the server side includes format:

```
<!--#var x--> <!--#var standard_html_footer-->
   <!--#var some_really_long_name--.
<!--#var and_another_rather_long_one-->
<!--#if input_name--> <!--#/if-->
```

Alternate Python String Format Syntax

This section describes the extended Python string format syntax. The extended Python string format syntax is used by the Python classes DocumentTemplate.String and DocumentTemplate.File. This format is based on the insertion-by-name format strings of Python with additional format characters, '[' and ']' to indicate block boundaries. In addition, attributes may be used within formats to control how insertion is done. For example:

```
%(date fmt=DayOfWeek upper)s
```

causes the contents of variable 'date' to be inserted using custom format 'DayOfWeek' and with all lower case letters converted to upper case.

Document template strings use an extended form of python string formatting. To insert a named value, simply include text of the form:

```
%(name)x
```

where `'name'` is the name of the value and `'x'` is a format specification, such as `'12.2d'`. To introduce a block such as an `'if'` or an `'in'` or a block continuation, such as an `'else'`, use '[' as the format specification. To terminate a block, use `']'` as the format specification, as in:

```
%(if input_name)[
   Hello %(var input_name size=16 etc="...")s.
%(elif nick_name)[
   Hi %(var nick_name capitalize)s.
%(else)[
   Have we been introduced?
%(/if)]
```

The form:

```
%(name)x
```

is a shorthand for :

```
%(var name)x
```

In most cases, the tag name, `'var'` can be omitted. It must be included when:

- The variable to be inserted is named `'var'` and

- when using the `expr` attribute:

```
%(var expr="foo+1")s
```

Common Tag Attributes

The previous section described two attributes that are used by most DTML tags, the *name* and *expr* attributes. Both of these attributes are used to identify or compute data used by the tag.

The *name* attribute

The *name* attribute is used to obtain data by name. The data is looked up using the rules described in the section "Name Lookup" below. The *name* attribute is special since there exists a shortened version of the attribute, as described in the section "DTML Tag Syntax" above.

When the value of a *name* attribute is looked up, the value is automatically called, if possible. If the value is a Zope Document or Python document template, it is rendered before being given to the tag that uses the *name* attribute. For example, most Zope documents begin with a *dtml-var* tag:

```
<dtml-var standard_html_header>
```

standard_html_header is a Zope document which provides standard HTML to be included at the top of every page. When the *var* tag above is used, the DTML in *standard_html_header* is rendered and the result is inserted in the current document.

If the value of a *name* attribute is a function[1] that is called with no arguments, then the result of the function call is given to the tag using the *name* attribute.

When the *name* attribute is used in the *dtml-if, dtml-elif, dtml-unless, dtml-in* or *dtml-with* tag, the value associated with the name is cached, causing references to the name in enclosed text to be faster than the initial reference. This is especially useful when the name refers to a function which is expensive to compute. For example:

```
<dtml-if reallyExpensiveFunction>
  <dtml-var reallyExpensiveFunction>
</dtml-if>
```

The *dtml-var* tag uses the cached value for `reallyExpensiveFunction`. Note that tags, such as *dtml-in* and *dtml-with*, which introduce new variables may introduce a new value for the given name, causing the cached value to be overridden.

The *expr* attribute

The *expr* attribute allows complex expressions to be evaluated. The expression used in an *expr* attribute is enclosed in double quotation marks. Thus for differentiation, double quotation marks are not allowed within the expression.

Expression syntax

The expression syntax is that of the Python[2] programming language and is similar to the syntax of other common programming languages like C or Java. Table 1 provides examples of simple expressions by giving the syntax and capabilities of each expression.

1. In this context, "function" refers any "callable" object. Examples of an callable objects include Zope Database Methods and Zope Network Clients, DTML Methods.
2. For a detailed description of Python syntax, see the Python documentation at *http://www.python.org/doc/*.

Expression	Explanation
`x*2+3`	Numeric expression
`func(a,b)`	Function call
`obj.title`	Get attribute `title` from `obj`
`obj.meth(a,b)`	Invoke method[i] `meth` of `obj` with arguments a and b
`(age < 12 or age > 65) and status == 'student'`	A boolean (true/false) test.
`REQUEST['HTTP_REFERER']`	Look up a value from `REQUEST` using the key `'HTTP_REFERER'`

TABLE 1. Expression examples

i. A method is like a function except, a method is an attribute of an object and can use the object's data in computation.

Variable lookup

The variable names used in an *expr* expression are looked up according to the rules described in "Name Lookup" Reference. Looked up values are not "called" as they are when the *name* attribute is used. In the expressions found in table 1, the name `x`, `func`, `a`, `b`, `obj`, `age`, and `status` are variable names, while the name `meth` and `title` are object attribute names.

Variable names must begin with a letter and contain only letters, digits, and underscore characters. To access a variable with a name that does not follow these rules, it is necessary to use the special variable, _, described in "The special namespace variable, _" below.

The special namespace variable, _

A special variable, _, is defined for use in *expr* expressions. The _ variable provides access to the DTML "namespace," which is an object used to look up variables when rendering DTML. The _ variable can be used to look up names directly:

```
<dtml-if "_['sequence-length'] > 20">
```

The _ variable is especially useful for accessing variables with names that contain special characters, like dashes in the case above.

The _ variable has a method, *has_key,* which can be used to check whether or not a variable is in the namespace:

```
<dtml-if "_.has_key('sequence-length')">
```

Note that when a callable value is looked up with the _ variable, it is called automatically, as is done with the name attribute. In the case where it is undesirable to have the value called automatically, the *getitem* method of the _ variable is used to look up the value. The *getitem* method accepts two arguments, the name to be looked up and a flag indicating whether or not the value is to be returned:

```
<dtml-var "_.getitem(name, 0)">
```

Name	Description
abs(X)	Return the absolute value of a number.
chr(I)	Return a string of one character whose ASCII code is integer `I`, e.g., `chr(97)` returns the string `'a'`. This is the inverse of `ord()`. The argument must be in the range 0-255, inclusive.
DateTime()	Create a DateTime object from zero or more arguments. See "Appendix A, Date-time data" on page 41.
divmod(A,B)	Take two numbers as arguments and return a pair of integers consisting of their integer quotient and remainder. With mixed operand types, the rules for binary arithmetic operators apply. For integers, the result is the same as`(A/B,A%B)`. For floating point numbers the result is the same as`(math.floor(A/B),A%B)`.
float(X)	Convert a number to floating point. The argument may be a plain or long integer or a floating point number.
getattr(O,name)	Get the named attribute from an object.
hasattr(O,name)	Test whether the name can be found in the namespace.
getitem(name,flag)	Lookup a name in the namespace. If the value is callable and the flag is true, then the result of calling the value is returned, otherwise the value is returned. flag defaults to false.
hash(O)	Return the 32-bit integer hash value of the object.
hex(X)	Convert an integer to a hexadecimal string.
int(X)	Convert a number to an integer.
len(S)	Return the length (the number of items) of a collection of items.
math	The math module (table 3), which defines mathematical functions.
max(S)	Return the largest item of a non-empty sequence.
min(S)	Return the smallest item of a non-empty sequence.
namespace(name1=value1, name2=value2, ...)	The *namespace* function can be used with the *in* tag to assign variables for use within a section of DTML. The function accepts any number of "keyword" arguments, which are given as `name=value` pairs.
None	A special value that means "nothing".
oct(X)	Convert an integer.
ord(C)	Return the ASCII value of a string of one character. E.g., `ord('a')` returns the integer `97`. This is the inverse of `chr()`.
pow(X,Y)	Return `X` to the power `Y`. The arguments must have numeric types. With mixed operand types, the rules for binary arithmetic operators apply. The effective operand type is also the type of the result; if the result is not expressible in this type, the function raises an exception, e.g. `pow(2,-1)` and `pow(2,35000)` raise exceptions.
random	The random module (table 4), which defines various random-number generating functions.
round(X,N)	Return the floating point value X rounded to N digits after the decimal point. If N is omitted, the default value is zero. The result is a floating point number. Values are rounded to the closest multiple of 10 to the power minus N; if two multiples are equally close, rounding is done away from 0 (so e.g. `round(0.5)` is `1.0` and `round(-0.5)` is `-1.0`).
str(O)	Return a string containing a representation of an object.
string	The string module, which defines string functions.
whrandom	The whrandom module (Figure 6, "Attributes defined by the whrandom module," on page 23), which defines random-number generating functions using the Wichmann-Hill pseudo-random number generator.

TABLE 2. Available attributes in the special variable, _

Name	Description
acos(X)	Compute the inverse cosine, in radians, of X.
asin(X)	Compute the inverse sine, in radians, of X.
atan(X)	Compute the inverse tangent, in radians, of X.
atan2(X,Y)	Compute the inverse tangent, in radians, of the quotient of X and Y.
ceil(X)	Return the smallest integer that is greater than X.
cos(X)	Compute the cosine of X, which is in radians.
cosh(X)	Compute the hyperbolic cosine of X.
e	The base of the natural logarithms.
exp(X)	Compute the exponential function of X, or e to the power X, where e is the base of the natural logarithms.
fabs(X)	Compute a floating-point absolute value of the number, X.
floor(X)	Return the largest integer less than X.
fmod(X,Y)	Return the remainder of the division of X by Y.
frexp(X)	Return the mantissa and exponent in base 2 of the floating-point value of X, such that absolute value mantissa is between 0.5 and 1.0 or is 0.
hypot(X,Y)	Compute the hypotenuse of a right triangle with sides X and Y.
ldexp(X,Y)	Return X times two to the power of Y.
log(X)	Compute the natural (base e) logarithm of X.
log10(X)	Compute the common (base 10) logarithm of X.
modf(X)	Break a number into its whole and fractional parts.
pi	The mathematical constant, pi
pow(X,Y)	Raise X to the power Y.
sin(X)	Compute the sine of X.
sinh(X)	Compute the hyperbolic sine of X.
sqrt(X)	Compute the square-root of X.
tan(X)	Compute the tangent of X.
tanh(X)	Compute the hyperbolic tangent of X.

TABLE 3. Attributes defined by the math module

Name	Description
betavariate (alpha, beta)	Beta distribution. Conditions on the parameters are alpha >- 1 and beta > -1. Returned values will range between 0 and 1.
choice (seq)	Choose a random element from the non-empty sequence seq and return it.
cunifvariate (mean, arc)	Circular uniform distribution. mean is the mean angle, and arc is the range of the distribution, centered around the mean angle. Both values must be expressed in radians, and can range between 0 and π. Returned values will range between mean - arc/2 and mean + arc/2.
expovariate (lambd)	Exponential distribution. lambd is 1.0 divided by the desired mean. (The parameter would be called "lambda", but that is a reserved word in Python.) Returned values will range from 0 to positive infinity.
gamma (alpha, beta)	Gamma distribution. (Not the gamma function!) Conditions on the parameters are alpha > -1 and beta > 0.
gauss (mu, sigma)	Gaussian distribution. mu is the mean, and sigma is the standard deviation. This is slightly faster than the normalvariate() function defined below.
lognormvariate (mu, sigma)	Log normal distribution. If you take the natural logarithm of this distribution, you'll get a normal distribution with mean mu and standard deviation sigma. mu can have any value, and sigma must be greater than zero.
normalvariate (mu, sigma)	Normal distribution. mu is the mean, and sigma is the standard deviation.
paretovariate (alpha)	Pareto distribution. alpha is the shape parameter.
randint (a, b)	Return a random integer N, such that a<=N<=b
random()	Return a random real number N, such that 0<=N<1.
uniform (a, b)	Return a random real number N, such that a<=N<b.
vonmisesvariate (mu, kappa)	mu is the mean angle, expressed in radians between 0 and pi, and kappa is the concentration parameter, which must be greater then or equal to zero. If kappa is equal to zero, this distribution reduces to a uniform random angle over the range 0 to .
weibullvariate (alpha, beta)	Weibull distribution. alpha is the scale parameter and beta is the shape parameter.

TABLE 4. Attributes defined by the random module

Name	Description
digits	The string '0123456789'.
hexdigits	The string '0123456789abcdefABCDEF'.
letters	The concatenation of the strings lowercase' and uppercase' described below.
lowercase	A string containing all the characters that are considered lowercase letters. On most systems this is the string 'abcdefghijklmnopqrstuvwxyz'.
octdigits	The string '01234567'.
uppercase	A string containing all the characters that are considered uppercase letters. On most systems this is the string 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
whitespace	A string containing all characters that are considered "white space". On most systems this includes the characters space, tab, line feed, return, form feed, and vertical tab.
atof(S)	Convert a string to a floating point number.
atoi(S[, BASE])	Convert string S to an integer in the given BASE. The string must consist of one or more digits, optionally preceded by a sign (+' or -'). The BASE defaults to 10. If the base is 0, a default base is chosen depending on the leading characters of the string (after stripping the sign): '0x' or '0X' means 16, '0' means 8, anything else means 10. If BASE is 16, a leading '0x' or '0X' is always accepted.
capitalize(W)	Capitalize the first character of the argument.
capwords(S)	Convert sequences of spaces, tabs, new-line characters, and returns to single spaces and convert every lower-case letter at the beginning of the string or that follows space to an uppercase letter.
find(S, SUB[, START])	Return the lowest index in S not smaller than START where the sub-string SUB is found. Return -1 when SUB does not occur as a sub-string of S with index at least START. If START is omitted, the default value is 0. If START is negative, then it is added to the length of the string.
rfind(S, SUB[, START])	Like find but find the highest index.
index(S, SUB[, START])	Like find but raise a ValueError exception when the substring is not found.
rindex(S, SUB[, START])	Like rfind but raise ValueError exception when the substring is not found.
count(S, SUB[, START])	Return the number of (non-overlapping) occurrences of substring SUB in string S with index at least START. If START is omitted, the default value is 0. If START is negative, then it is added to the length of the string.

TABLE 5. Attributes defined by the string module

Name	Description
lower(S)	Convert letters to lower case.
maketrans(FROM, TO)	Return a translation table suitable for passing to string.translate, that will map each character in FROM into the character at the same position in TO; FROM and TO must have the same length.
split(S[, SEP[, MAX]])	Return a list of the words of the string S. If the optional second argument SEP is absent or None, the words are separated by arbitrary strings of white-space characters (space, tab, new line, return, form feed). If the second argument SEP is present and not None, it specifies a string to be used as the word separator. The returned list will then have one more items than the number of non-overlapping occurrences of the separator in the string. The optional third argument MAX defaults to 0. If it is nonzero, at most MAX number of splits occur, and the remainder of the string is returned as the final element of the list (thus, the list will have at most MAX+1 elements).
join(WORDS[, SEP])	Concatenate a list or tuple of words with intervening occurrences of SEP. The default value for SEP is a single space character. It is always true that string.join(string.split(S, SEP), SEP) equals S.
lstrip(S)	Remove leading white space from string S.
rstrip(S)	Remove trailing white space from string S.
strip(S)	Remove leading and trailing white space from string S.
swapcase(S)	Convert lower case letters to upper case and vice versa.
translate(S, TABLE[, DELS])	Delete all characters from S that are in DELS (if present), and then translate the characters using TABLE, which is a 256-character string giving the translation for each character value, indexed by its ordinal.
upper(S)	Convert letters to upper case.
ljust(S, WIDTH) *rjust(S, WIDTH)* *center(S, WIDTH)*	These functions respectively left-justify, right-justify and center a string in a field of given width. They return a string that is at least WIDTH characters wide, created by padding the string S with spaces until the given width on the right, left or both sides. The string is never truncated.
zfill(S, WIDTH)	Pad a numeric string on the left with zero digits until the given width is reached. Strings starting with a sign are handled properly.

TABLE 5. Attributes defined by the string module

Name	Description
choice (seq)	Choose a random element from the non-empty sequence seq and return it.
randint (a, b)	Return a random integer N, such that a<=N<=b
random()	Return a random real number N, such that 0<=N<1.
seed(X, Y, Z)	Initialize the random number generator from the integers X, Y and Z.
uniform (a, b)	Return a random real number N, such that a<=N<b.

TABLE 6. Attributes defined by the whrandom module

Name Lookup

When a variable name is used in a DTML tag, such as a *dtml-var* tag or an *expr* attribute expression, that name must be resolved. Table 7 shows the steps taken to look up data in the simplest case.

Step	Rule
1	When a document template is called from Python, the mapping object and keyword arguments supplied when calling the document template are searched, with keyword arguments taking precedence over the mapping object.
2	When a document templates is called from Python, attributes, including inherited and acquired attributes, of the client passed in the call to the document template are searched.
3	If DTML is used in a Zope *DTML Method or Document* and the variable name is `document_id` or `document_title`, then the id or title of the document is used.
4	Attributes of the folder containing the DTML are searched. Attributes include objects in the contents of the folder, properties of the folder, and other attributes defined by Zope[i], such as *ZopeTime*[ii]. Folder attributes include the attributes of folders containing the folder, with contained folders taking precedence over containing folders.
5	Search Zope-defined Web-request data (table 8).
6	Search variables defined in Web-request form data.
7	Search variables defined in Web-request cookies.
8	Search variables named *URLn*, where n is a digit. *URL0* is the Request URL without a query string. *URL1* is the same as *URL0* except that the last name in the URL is removed. *URL2* is the same as URL0 except that the last two names are removed, and so on. For example, if the request URL is `http://digicool.com/A/B`, then *URL0*, *URL1*, and *URL2* are respectively, `http://digicool.com/A/B`, `http://digicool.com/A` and `http://digicool.com`. *URL3* is undefined.
9	Search CGI-defined Web-request variables. See table 11 for a description of CGI-defined variables.
10	Search HTTP Headers. A variable name associated with a HTTP header consist of the HTTP header name, converted to upper case, with the Prefix, *HTTP_*. For example, a HTTP Referer header, if present, can be accessed using the variable name *HTTP_REFERER*.
11	Search variables named *BASEn*, where n is a digit. *BASE0* is the prefix of the request URL up to, but not including, the name of the Zope installation or module published by ZPublisher. *BASE1* is the request URL up to and including the name of the Zope installation or module published by ZPublisher. *BASE2* is the request URL up to the name following the name of the Zope installation or module published by ZPublisher, and so on. For example, assume that a Zope installation or module published by ZPublisher has the URL: `http://digicool.com/Demos/Plutonia` and that a request URL is `http://digicool.com/Demos/Plutonia/Marketing`. *BASE0* is `http://digicool.com/Demos`, *BASE1* is `http://digicool.com/Demos/Plutonia` and *BASE2* is `http://digicool.com/Demos/Plutonia/Marketing`. *BASE3* is undefined.

TABLE 7. Simplest-case steps for looking up names

i. Zope defines a large number of attributes that are used by Zope itself. Many of these will become part of an official Zope applications programming interface and will be documented in a forthcoming "Zope API Guide".

ii. *ZopeTime* is a function that returns a date-time object giving the current time.

There are two situations in which the search rules for the simplest case are modified. If a Zope object or Python document template is called within a DTML *expr* attribute expression, then additional variables may be passed in. Variables passed in take precedence over all variables described in Table 6.

Name	Description
AUTHENTICATED_USER	An object that represents an authenticated user. When inserted into a DTML document, the value is the user name. This object *currently* provides no public attributes. Note that this variable may not be defined in Documents that are not protected by security information.
AUTHENTICATION_PATH	The path to the object containing the user database folder which contained the definition for the authenticated user.
PARENTS	A sequence of ancestors of the object that was accessed in the current request. For example, if the accessed object is a Document, then PARENTS[0] is the folder containing the document, PARENTS[1] is the folder containing the folder containing the document, and so on.
REQUEST	An object that represents the current request. This object may be used in an *expr* expression to look up request data, including variables described in this table, CGI-defined variables (table 11), form variables, cookies, and HTTP headers. In addition, *expr* expressions may use request attributes defined in table 9
RESPONSE	An object that represents the response to the current request. This object is primarily useful in *expr* expressions using attributes defined in table 10.
URL	The URL used to invoke the request without the query string, if any.

TABLE 8. Zope-defined Web request variables

Name	Description
cookies	If a HTTP Cookie was included in the request, then this attribute is a dictionary[i] containing the cookie data. This allows cookie data to be looked up, even if a cookie name is the same as a form variable or an object attribute name.
form	If a request was initiated by submitting a form, then the form attribute is a dictionary[a] containing the form data. This allows form data to be looked up, even if a form name is the same as an object attribute name.
has_key(name)	Determine whether the REQUEST defines a given name.
set(name, value)	Set a variable in the request.

TABLE 9. Attributes of the REQUEST variable.

i. Dictionaries are objects that support looking of data by name (e.g. `REQUEST.cookies['CUST_ID']` to look up a cookie named `CUST_ID`). Dictionaries have *has_key* methods for checking whether a dictionary contains a value (e.g. `REQUEST.cookies.has_key('CUST_ID')`) and methods *keys*, *values*, and *items*, for updating lists of dictionary keys, values, and key-value pairs (e.g. `REQUEST.cookies.keys()` to obtain a list of cookie names).

Some DTML tags define additional variables. Variables defined by DTML tags take precedence over variables described in table 6. If tags are nested, variables defined in nested tags take precedence over variables defined in tags that are nested in.

Names may not begin with an underscore, except in the special case of the _variable used in an *expr* attribute expression.

If a variable lookup yields an object that has security information, then access to the variable is allowed only if the user on whose behalf the DTML is being rendered is allowed to access the object.

Name	Description
setStatus(status)	Set the HTTP status code of the response; the argument may either be an integer or a string from {OK, Created, Accepted, NoContent, MovedPermanently, MovedTemporarily, NotModified, BadRequest, Unauthorized, Forbidden, NotFound, InternalError, NotImplemented, BadGateway, ServiceUnavailable} that will be converted to the correct integer value.
setHeader(name, value)	Set a HTTP return header `name` with value, clearing the previous value set for the header, if one exists.
getStatus()	Return the current HTTP status code as an integer.
setBase(base)	Set the base URL for the returned document.
expireCookie(name,...)	Cause a HTTP cookie to be removed from the browser The response will include a HTTP header that will remove the cookie corresponding to “name” on the client, if one exists. This is accomplished by sending a new cookie with an already passed expiration date.
setCookie(name,value,...)	Cause the response to include a HTTP header that sets a cookie on cookie-enabled browsers with a key `name` and value. This overwrites any previously set value for the cookie in the Response object. Additional cookie parameters can be included by supplying keyword arguments. The valid cookie parameters are *expires*, *domain*, *path*, *max_age*, *comment*, and *secure*.
getHeader(name)	Return the value associated with a HTTP return header or None if no such header has been set in the response.
appendHeader(name, value)	Set a HTTP return header “name” with value “value” and appending it following a comma if there is a previous value set for the header.
redirect(location)	Cause a redirection without raising an error.

TABLE 10. Attributes of the RESPONSE variable

Name	Description
SERVER_SOFTWARE	The name and version of the information server software answering the request. Format: name/version
SERVER_NAME	The server's host name, DNS alias or IP address as it would appear in self-referencing URLs.
GATEWAY_INTERFACE	The revision of the CGI specification to which this server complies. Format: CGI/revision.
SERVER_PROTOCOL	The name and revision of the information protocol this request came in with. Format: protocol/revision
SERVER_PORT	The port number to which the request was sent.

TABLE 11. CGI-defined Web request variables

Name	Description
REQUEST_METHOD	The method with which the request was made. For HTTP, this is "GET", "HEAD", "POST", etc.
PATH_INFO	The part of the request URL, not counting the query string, following the name of the Zope installation or module published by ZPublisher.
PATH_TRANSLATED	The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.
SCRIPT_NAME	A virtual path to the script being executed, used for self-referencing URLs.
QUERY_STRING	The information which follows the ? in the URL which referenced this script. This is the query information.
REMOTE_HOST	The host name making the request. If the server does not have this information, it should REMOTE_ADDR and leave REMOTE_HOST unset.
REMOTE_ADDR	The IP address of the remote host making the request.
AUTH_TYPE	If the server supports user authentication, and the script is protected, this is the protocol-specific authentication method used to validate the user.
REMOTE_USER	If the server supports user authentication, and the script is protected, this is the username under which it has been authenticated.
REMOTE_IDENT	If the HTTP server supports RFC 931 identification, then this variable will be set to the remote username retrieved from the server. Usage of this variable should be limited to logging only.
CONTENT_TYPE	For queries which have attached information, such as HTTP POST and PUT, this is the content type of the data.
CONTENT_LENGTH	The length of the said content as given by the client.

TABLE 11. CGI-defined Web request variables

Access Control

Document templates provide a basic level of access control by preventing access to names beginning with an underscore[1]. Additional control may be provided by providing document templates with a 'validate' method. This would typically be done by subclassing one or more of the DocumentTemplate classes.

If provided, the 'validate' method will be called when objects are accessed as instance attributes or when they are accessed through keyed access in an expression. The 'validate' method will be called with five arguments:

1. The containing object that the object was accessed from,
2. The actual containing object that the object was found in, which may be different from the containing object the object was accessed from, if the containing object supports acquisition,
3. The name used to access the object,
4. The object, and
5. The name-space object used to render the document template.

If a document template is called from Zope, then the name-space object will have an attribute AUTHENTICATED_USER that is the user object that was found if Zope authenticated a user.

1. The special variable, _, is an exception to this rule.

Zope document objects provide their own validate method that implements the security rules of the Zope application framework.

Using Document Templates from Python

Document templates are made available using the DocumentTemplate package[1]. The DocumentTemplate package defines four classes to be used depending on whether source is stored in Python strings or in external files and on whether the HTML server-side include syntax or the extended Python string format syntax is used. The four document template classes are shown in table 11.

Class name	Description
DocumentTemplate.HTML	Source is provided as a Python string in HTML server-side-include syntax.
DocumentTemplate.HTMLFile	Source is provided as an external file in HTML server-side-include syntax.
DocumentTemplate.String	Source is provided as a Python string in extended Python string format syntax.
DocumentTemplate.File	Source is provided as an external file in extended Python string format syntax.

TABLE 12. Document template classes

Creating document templates

Document templates are created by calling one of the classes listed in table 11. The source is passed as the first argument. An optional mapping argument may be provided that contains names to be added to the document template namespace when called and default values. An optional third argument may be provided to specify a namespace attribute for the document template. The standard document template creation arguments are listed in table 12.

Argument number	Argument name	Description
1	source_string or file_name	The document template source. For classes String and HTML, this must be a string. For classes File and HTMLFile, this is the name of an external file.
2	mapping	A mapping object containing initial names and default values for the document template name-space.
3	__name__	A value to use for the __name__ attribute of the document template object. This value can be used by programming tools to identify the object. For example, Zope tracebacks include __name__ values for instances of methods listed in tracebacks. For classes String and HTML, the default __name__ value is '<string>' and for File and HTMLFile classes, the default value is the file name.

TABLE 13. Standard document template creation arguments.

In addition to the standard creation arguments, additional keyword arguments may be provided to provide additional names and default values for the document template name-space. For example, in:

```
results=DocumentTemplate.HTMLFile('results.dtml',
  {'table_name': 'search results', 'database': 'data'},
  pid=os.getpid(),
  time=time.time
  )
```

1. Python 1.4 users must use the ni module to enable packages. DocumentTemplate may also be used as a collection of modules, rather than as a package by copying all of the DocumentTemplate modules except the __init__ module to a directory in the Python path.

A document template is created using server-side-include syntax source from an external file named ’results.dtml’ and with an initial name space that included the names ’table_name’, ’database’, ’pid’, and ’time’.

Using document templates

To generate text using a document template, the document template must be called. Arguments may be provided to supply an object from which to get data, a mapping object to get data from, or keyword arguments containing additional data. The standard arguments for calling document templates are shown in table 13.

Argument number	Argument name	Description
1	client	An object with attributes to be included in the document template namespace.
		The client may be a single object, or a tuple of objects. If client is a tuple, then each object will be added to the document template namespace in order.
2	mapping	A mapping object with names and values to be added to the namespace.

TABLE 14. Standard arguments for calling document templates.

Both of the standard arguments may be omitted. If the mapping argument is to be provided positionally without a client, then a None must be passed as the client value, as in:

```
return results(None, {’search_results’: r})
```

Keyword arguments may be used to provide values, as in:

```
return results(search_results=r)
```

Using document templates with ZPublisher

Document templates may be published directly with ZPublisher. ZPublisher treats document templates as if they were Python functions that accept the arguments named `’self’` and `’REQUEST’`. The object traversed to get to the document template is passed as the `’self’` argument and the request object is passed as the `’REQUEST’` argument. Typically, document templates are defined as class attributes and passed class instances and request data, so instance and request data can be used to generate text.

Document templates may also be used in Python functions called by document templates, as in:

```
def getResults(self, key, REQUEST):
   result_data=self.search(key)
   return self.resultTemplate(selfdocument, REQUEST,
search_results=result_data)
```

Be sure to call the document template. A common mistake is to return the document template directly, as in:

```
def getResults(self, key, REQUEST):
   result_data=self.search(key)
   return self.resultTemplate
```

ZPublisher does not attempt to call an object returned from a published object. Results of calling a published function are simply converted to strings and returned. When a document template is converted to a string, the document template source is returned. In the example above, the document template source, rather than the rendered template is returned.

The *dtml-var* Tag

The *dtml-var* tag is used to perform simple variable substitutions. A number of attributes are provided to control how text is to be inserted and formatted. The attributes are summarized in table 15.

Attribute name	Needs an argument?	Description
name	yes	Insert the name of the variable.
expr	yes	Insert an expression that evaluates a value.
fmt	yes	Specify a data format, which may be a special, custom, or C-style format.
null	yes	Specify a string to be substituted for null values.
lower	no	Convert all upper-case letters to lower case
upper	no	Convert all lower-case letters to upper case.
capitalize	no	Convert the first character of the inserted to upper case.
spacify	no	Convert underscores in the inserted value to spaces.
thousands_commas	no	In a value containing all numbers, insert commas every three digits to the left of a decimal point. For example, "12000 widgets" becomes "12,000 widgets".
html_quote	no	Convert characters that have special meaning in HTML to HTML character entities.
url_quote	no	Convert characters that have special meaning in URLs to HTML character entities using decimal values.
url_quote_plus	no	Converts a single space character in URLs to the plus sign "+".
sql_quote	no	Convert single quotation mark to a pair of single quotation marks. This is needed to safely include values in Standard Query Language (SQL) strings.
newline_to_br	no	Convert new-line characters, carriage-return characters, and new-line-carriage-return character combinations to new-line characters followed by HTML break tags.
size	yes	Truncate the value to the given size.
etc	yes	Provide a string to be added to truncated text to indicate that truncation has occurred. The default value is "...".
missing	no	Provide a value to be used if the variable is missing, rather than raising an KeyError. Used without an argument, it will provide an empty string.

TABLE 15. *dtml-var* tag attributes

Custom, Special, C, and Empty Formats

A custom format is used for the output of objects. The value of a custom format is the method name evaluated upon the object to be inserted. The method should return an object that, when converted to a string, yields the desired text. For example, the DTML source text:

```
<dtml-var date fmt=DayOfWeek>
```

inserts the result of calling the method, `DayOfWeek`, with the value of the variable `date`. As the example suggests, the most common use of custom formats in Zope is in the output of date-time data. The appendix A provides a summary of the custom formats available for date-time data.

In addition to custom formats, a few special formats are defined by the *var* tag that can be used with the *fmt* attribute. These are summarized in table 16.

Special Format	Description
whole-dollars	Show a numeric value with a dollar symbol.
dollars-and-cents	Show a numeric value with a dollar symbol and two decimal places.
collection-length	Get the length of a collection of objects.
structured-text	Allows use of structured text as a special format.

TABLE 16. Special formats that may be used with the *var* tag *fmt* attribute

In addition to custom and special formats, C[1]-style formats may also be used. A C-style format consists of text containing a single conversion specification. A conversion specification consists of a percent sign, optionally followed by a flag, a field width, a precision value, and a conversion specifier. A description of C-style formats is beyond the scope of this document. For details on C-style formats, see a C-language reference manual. Not all conversion specifiers are supported by DTML. Table 17 summarizes the conversion specifiers that DTML does support.

Code	Description
d	Signed decimal integers
e	Scientific notation
E	Scientific notation (uppercase E)
f	Decimal floating point
g	Shorter of e or f
G	Shorter of E or F
I	Signed decimal integers
o	Unsigned octal
s	String of characters
u	Unsigned decimal integers
x	Unsigned hexadecimal lowercase
X	Unsigned hexadecimal uppercase

TABLE 17. C-style specifiers for the *fmt* attribute

Null Values

In some applications, and especially in database applications, data variables may alternate between "good" and "null" or "missing" values. A format that is used for good values may be inappropriate for null values. For this reason, the *null* attribute can be used to specify text to be inserted for null values. Null values are defined as values which:

- Cannot be formatted with the specified format, and
- Are either the special Python value `None` or are false and yield an empty string when converted to a string.

1. The C programming language.

For example, when showing a monetary value retrieved from a database that is either a number or a missing value, the following variable insertion might be used:

```
<dtml-var cost fmt="$%.2d" null='n/a'>
```

Truncation

The attributes *size* and *etc* can be used to truncate long strings. If the *size* attribute is specified, then the string to be inserted is truncated at the given length. If a space occurs in the second half of the truncated string, then the string is further truncated to the right-most space. After truncation, the value given for the *etc* attribute is added to the string. If the *etc* attribute is not provided, then "`...`" is used. For example, if the value of the variable *color* is "red yellow orange green blue", then the tag:

```
<dtml-var spam size=10 etc="...">
```

inserts

```
red yellow...
```

A *dtml-var* Tag Example, the Default Document Source

When a Zope Document is created and no source is given, Zope supplies a default DTML source, as shown in figure 1.

```
<dtml-var standard_html_header>
<H2><dtml-var title_or_id> <dtml-var document_title></H2>
<P>This is the <dtml-var document_id> Document in
the <dtml-var title_and_id> Folder.</P>
<dtml-var standard_html_footer>
```

Figure 1. Default Zope Document source

In figure 1, several *dtml-var* tags are used. The first *var* tag inserts *standard_html_header*. Inserted in almost all Zope Documents, the *standard_html_header* is a Document which provides a standard way to begin HTML documents in a Zope installation. Thus, the inserted Document gives a Zope installation a common “look and feel”. The *standard_html_header* document can be edited to customize a Zope installation and can be overridden in sub-folders to give different parts of a site varying appearances. The document *standard_html_footer* provides a similar function for the end of Zope documents. The *var* tags that insert *standard_html_header* and *standard_html_footer* illustrate the notion that DTML documents can be called from another DTML document.

The variables *title_or_id* and *title_and_id* are methods defined on most Zope objects. The method *title_or_id* returns the object’s title if the title is not blank, otherwise the object’s id is returned. The method *title_and_id* returns the title of an object followed by the id in parentheses if the title is not blank, otherwise the id is returned. In figure 1, the *title_or_id* and *title_and_id* methods are applied to the folder containing the document. The example illustrates the use of *var* tags to insert the results of method calls.

The variables *document_id* and *document_title* simply return the id and title of the document.

Conditional Insertion, the *dtml-if* and *dtml-unless* Tags

Occasionally, the text to be included in a document is dependent upon some data. The *if* tag is provided to support the conditional insertion of text based on DTML variables or expressions. As described in “DTML Tag Syntax”, the *dtml-if* tag has four forms:

1. An *dtml-if* tag, with a closing */dtml-if* tag,
2. An *dtml-if* tag with an *dtml-else* tag and a closing */dtml-if* tag.
3. An *dtml-if* tag with one or more *dtml-elif* tags, an *dtml-else* tag, and a closing */dtml-if* tag, and
4. An *dtml-if* tag with one or more *dtml-elif* tags, no *dtml-else* tag, and a closing */if* tag.

The *dtml-if* tag works in a straightforward manner. The variable or expression given in the *dtml-if* tag is evaluated. If the variable or expression value is true[1], then the text following the *dtml-if* tag is inserted. If the variable or expression value is false, then for each *dtml-elif* tag given, the variable or expression given in the *dtml-elif* tag is evaluated. If an *dtml-elif* variable or expression value is true, the text following the *dtml-elif* tag is inserted and none of the following *dtml-elif* variables or expressions are evaluated. If there are no *dtml-elif* tags or if all of the *dtml-elif* tag variable or expression values are false, then the text following the *dtml-else* tag is inserted. If no *dtml-else* tag was supplied, then no text is inserted.

The *dtml-if* and *dtml-elif* tags support only the standard *name* and *expr* attributes. The *dtml-else* tag accepts no attributes.

In addition to the *dtml-if* tag, the *dtml-unless* tag is provided with its associated closing tag, */dtml-unless*, to insert text if a condition is false. Like the *dtml-if* tag, the *dtml-unless* tag accepts the standard *name* and *expr* attributes:

```
<dtml-unless input_name>
  You did not provide a name.
</dtml-unless>
```

1. All Zope objects are either true or false. Numeric values are true if they are non-zero and false if they are zero. Objects that are sequences of objects, like search results, are true if the sequences are non-empty and false otherwise. Most other objects are true.

Iterative Insertion, the *dtml-in* Tag

Commonly, it is necessary to insert a sequence of values. Some objects, like *Zope SQL Method*s, and *Confera Topics* support searching and a means is needed for iterating over search results.

When creating input forms with select lists, it is sometimes a good idea to store the contents of the list in a folder property so that the list can be edited independently from the input form. In this case, the select list options are inserted into a form by iterating over the list property.

The *dtml-in* tag is used to iterate over a sequence of objects. For example, an employee directory listing might be created with DTML source like that shown in figure 2. In this example, `employees` is either a sequence of employees, or a function, such as an *Zope SQL Method*, that computes a sequence of employees. Each employee has a `name` and `phone` attribute. These attributes are accessed using *dtml-var* tags. An *dtml-in* tag's *sort* attribute is used to sort employees by name in the output. The *dtml-in* tag attributes are listed in table 18.

```
<table>
  <tr><th>Name</th><th>Phone number</th></tr>
  <dtml-in employees sort=name>
    <tr>
      <td><dtml-var name></td>
      <td><dtml-var phone></td>
    </tr>
  </dtml-in>
</table>
```

Figure 2. DTML source to create an employee phone listing

In the example, an empty table would be displayed if there were no employees. To avoid displaying an empty table a message can be provided indicating that there are no employee by using the *in* tag in combination with the *dtml-if* tag, as shown in figure 3. In figure 3, the *dtml-if* tag is used with the `employees` variable. Sequences of objects are false if they are empty and true if they are not. If there are no employees, the condition in the *dtml-if* tag is false and the text following the *dtml-else* tag is inserted.

The *dtml-else* Tag as an Intermediate Tag in the *dtml-in* Tag

In the previous example (figure 2) an *dtml-in* tag is combined with an *dtml-if* tag to avoid showing an empty table for an empty sequence of employees. An alternative approach is to use an intermediate *dtml-else* tag in the *dtml-in* tag. If an *dtml-in* tag has an intermediate *dtml-else* tag, then the text following the *dtml-else* tag is inserted if the sequence used in the *dtml-in* tag is empty. Figure 4 shows DTML source which uses an *dtml-else* tag in the *dtml-in* tag to avoid showing an empty table. The output from this source is the same as the output from the source shown in figure 3. The source in figure 4 is actually more complex that the source in figure 3. The added complexity is due to the fact that the table header and footer have to be moved inside the *dtml-in* tag. Furthermore, the insertion of the table header and footer has to be conditioned on whether or not an item is the first item, last item, or neither by using the variables *sequence-start* and *sequence-end* (table 19).

Name	Needs an argument	Description
name	yes	Insert the name of the variable. See "The name attribute".
expr	yes	Insert an expression that evaluates the value. See The expr attribute".
mapping	no	Normally, the attributes of items in the sequence are displayed. But, some items should be treated as mapping objects, meaning that the items are to be looked up.
sort	yes	The sort attribute is used to cause a sequence of objects to be sorted before text insertion is performed. The attribute value is the name of the attribute (or key if the mapping attribute was provided) that items should be sorted on.
start	yes	The name of a (request) variable that specifies the number of the row on which to start a batch.
size	yes	The batch size.
skip_unauthorized	no	Use of this attribute causes items to be skipped if access to the item is unauthorized. See "Access Control". If this attribute is not used, then Unauthorized errors are raised if unauthorized items are encountered.
orphan	yes	The desired minimum batch size.
overlap	yes	The number of rows to overlap between batches.
previous	no	If the previous attribute is included, then iterative insertion will not be performed. The text following the *in* tag will be inserted and batch processing variables associated with information about a previous batch will be made available.
reverse	no	Used with the sort attribute, objects can be sorted in reverse order.
next	no	The next attribute has the same meaning and use as the previous attribute except that variables associated with the next batch are provided.

TABLE 18. *dtml-in* tag attributes

```
<dtml-if employees>
  <table>
    <tr><th>Name</th><th>Phone number</th></tr>
    <dtml-in employees sort=name reverse>
      <tr>
        <td><dtml-var name></td>
        <td><dtml-var phone></td>
      </tr>
    </dtml-in>
  </table>
<dtml-else>
  Sorry, there are no employees.
</dtml-if>
```

Figure 3. DTML source to create an employee phone listing which properly handles the case of no employees by using an *in* tag with an *dtml-if* tag.

When a *name* attribute is used in an *dtml-in* tag within an *if* tag, the sequence is only evaluated once, since the *dtml-if* tag caches the value associated with a *name* attribute.

```
<dtml-in employees sort=name>
  <dtml-if sequence-start>
    <table>
      <tr><th>Name</th><th>Phone number</th></tr>
  </dtml-if>
  <tr>
      <td><dtml-var name></td>
      <td><dtml-var phone></td>
  </tr>
  <dtml-if sequence-end>
    </table>
  </dtml-if>
<dtml-else>
  Sorry, there are no employees.
</dtml-in>
```

Figure 4. DTML source to create an employee phone listing which properly handles the case of no employees by using an dtml-else tag in an dtml-in tag.

In most cases, it is best to use an *dtml-in* tag inside an *dtml-if* tag, as illustrated in figure 3. One case in which it may be best to use an *dtml-else* tag within an *dtml-in* tag is when the sequence used by the *dtml-in* tag is computed using an *expr* attribute and the computation is expensive. Use of the *dtml-else* tag in the *dtml-in* tag avoids having to define and evaluate the expression twice.

Variables Defined by the *dtml-in* Tag

When text is inserted using an *dtml-in* tag, a copy of the text is inserted for each item in the sequence. Tags in the inserted text have access to variables not available outside the *dtml-in* tag. These include:

- Attributes of the current item,
- Item variables defined by the *dtml-in* tag (table 19),
- Summary statistic variables defined by the *dtml-in* tag (table 19),
- Grouping variables defined by the *dtml-in* tag (table 21), and
- Batch-processing variables defined by the *dtml-in* tag (table 22).

In addition, for each of the variables listed in tables 19, 22, and 23 with names ending in "-index", there exist variables whose names end in "-number", "-roman", "-Roman", "-letter", and "-Letter" which are indexed from 1, "i","I", "a", and "A", respectively. The *sequence-index* variable is used to number items as text is inserted. Variables like *sequence-letter* and *sequence-roman* provide numbering using letters and Roman numerals.

There also exist variables ending in "-even" and "-odd", which test whether the sequence index is even or odd. This is useful to display rows more visibly by alternating colors.

Name	Description
sequence-item	The current item.
sequence-key	The key associated with the element in an items sequence. An items sequence is a sequence of value pairs that represent a mapping from a key to a value.
sequence-index	The index, starting from 0, of the element within the sequence.
sequence-start	The variable is true if the element being displayed is the first of the displayed elements, and false otherwise. This is useful when text must be inserted at the beginning of an *dtml-in* tag, especially if the text refers to any variables defined by the *dtml-in* tag.
sequence-end	The variable is true if the element being displayed is the last of the displayed elements, and false otherwise. This is useful when text must be inserted at the end of an *dtml-in* tag, especially if the text refers to any variables defined by the *dtml-in* tag.

TABLE 19. Item variables defined by the *dtml-in* tag

Finally, for each of the variables ending in "-index", there are variables whose names end in "-var-xxx", where "xxx" is an element attribute name or key. This is useful when displaying previous- and next-batch information. The construct is also useful if used in an *dtml-if* tag to test whether or not an attribute is present since the attribute lookup will not be extended to the full document template name space.

Summary Statistics

The *dtml-in* tag provides variables (table 20) for accessings. Summary statistics are computed over the entire sequence, not just over the items displayed.

Name	Description
total-nnn[i]	The total of numeric values.
count-nnn	The total number of non-missing values.
min-nnn	The minimum of non-missing values.
max-nnn	The maximum of non-missing values.
median-nnn	The median of non-missing values.
mean-nnn	The mean of numeric values.
variance-nnn	The variance of numeric values computed with a degrees of freedom equal to the (count - 1).
variance-n-nnn	The variance of numeric values computed with a degrees of freedom equal to the count.
standard-deviation-nnn	The standard deviation of numeric values computed with a degrees of freedom equal to the (count - 1).
standard-deviation-n-nnn	The standard deviation of numeric values computed with a degrees of freedom equal to the count.

TABLE 20. Summary statistic variables defined by the *dtml-in* tag

i. nnn is the name of an attribute or key. For example, to get the mean salary in a collection of employees, each with the attribute `salary`, `mean-salary` would be used.

Grouping Variables

The *dtml-in* tag defines special variables used for testing when a "grouping" variable changes. These variables begin with the prefix "first-" or "last-". Their value is used to test whether an item is the first or last item in a subsequence of displayed items whose value is the same value for the given item variable.

Name	Description
first-nnn	True if the current item is the first item among the displayed items that has the current value for variable, `nnn`; False otherwise.
last-nnn	True if the current item is the last item among the displayed items that has the current value for variable, `nnn`; False otherwise

TABLE 21. Special variables for group processing

Batch Processing

When displaying a large number of objects, it may be impractical to display all of the data at once. While the approach used in figure 3 is practical for a small group of employees, it is impractical for browsing the employees of a large company.

For this reason, the *dtml-in* tag provides support for batch processing. Information is displayed in batches. Variables are provided (table 22) to aid in the construction of HTML hyperlinks to other batches.

Name	Description
sequence-query	The original query string given in a get request with the form variable named in the *start* attribute removed.
sequence-step-size	The batch size used.
previous-sequence	The variable is true when the first element is displayed, and when the first element displayed is not the first element in the sequence.
previous-sequence-start-index	The index, starting from 0, of the start of the batch previous to the current batch.
previous-sequence-end-index	The index, starting from 0, of the end of the batch previous to the current batch.
previous-sequence-size	The size of the batch previous to the current batch.
previous-batches	A sequence of **mapping** objects containing information about all of the batches prior to the batch being displayed.
next-sequence	The variable is true when the last element is displayed, and when the last element displayed is not the last element in the sequence.
next-sequence-start-index	The index, starting from 0, of the start of the batch after the current batch.
next-sequence-end-index	The index, starting from 0, of the end of the batch after the current batch.
next-sequence-size	The size of the batch after the current batch.
next-batches	A sequence of **mapping** objects containing information about all of the batches after the batch being displayed.

TABLE 22. Batch-processing variables

Name	Description
batch-start-index	The index, starting from 0, of the beginning of the batch.
batch-end-index	The index, starting from 0, of the end of the batch.
batch-size	The size of the batch.

TABLE 23. Attributes of batch objects used when iterating over *next-batches* and *previous-batches* variables.

The batch-processing facilities of the *dtml-in* tag are quite powerful, but the various options and approaches are complex. For lucidity, take for example a simple table of 36 words (figure 5). The DTML source in figure 6 is used to display this data. The DTML uses an *if* tag to test for an empty sequence of words. The actual sequence is named `w36`. Inside the *if* tag, there are three *dtml-in* tags. All three *dtml-in* tags include the attributes, *size* with the value 5 and *start* with the value `qs`. The *size* attribute is used to specify a batch size. For example purposes, the batch size is unusually small. The *start* parameter is used to specify the name of a variable which holds the index of the first element of the sequence to be displayed. If the variable is not defined, then the first batch is displayed. Figure 7 shows the output of the DTML as displayed on a Web browser for the first two and last two batches.

Number	word
1	accident
2	assault
3	assert
4	bask
5	berlin
6	berlin
7	buttress
8	center
9	clamor
10	distort
11	envelop
12	extend

TABLE 24.

Number	word
13	eye
14	fool
15	graft
16	index
17	jet
18	lull
19	market
20	marshal
21	mask
22	meet
23	neck
24	offer

TABLE 24.

Number	word
25	output
26	peak
27	ration
28	reprogram
29	school
30	sear
31	sex
32	shake
33	slam
34	spawn
35	trivial
36	vital

TABLE 24.

Figure 5. Table of 36 words

The first of the three *dtml-in* tags is used to display an HTML hyperlink to a previous batch of data. The *previous* attribute in the *dtml-in* tag indicates that only previous-batch data should be displayed. Row data are not displayed. If the first batch is being displayed, then no text is inserted (figure 7 (words 1-5)). The source in the first *dtml-in* tag uses four variable references. The first retrieves the *document_id*, which is used as a relative URL name for the document. The second variable reference uses *sequence-query* to retrieve the request query string which has been modified so that it does not include the variable named in the *dtml-in* tag *start* attribute, `qs`. The *sequence-query* value also contains the necessary punctuation, '`?`' and '`&`', so that the *document_id*, *sequence-query* and URL-encoded value for the next batch start can be concatenated. The URL-encoded value of the next batch start is "`qs=`" followed by the variable, *previous-sequence-start-number*. The variable *previous-sequence-size* provides the size of the previous batch for display in the hyperlink. Note that the previous (or next) sequence size is not necessarily equal to the batch size.

The DTML source has been split over multiple lines by introducing line breaks within *var* tags. This is a useful way to break up long lines without causing line-breaks to be included in generated HTML.

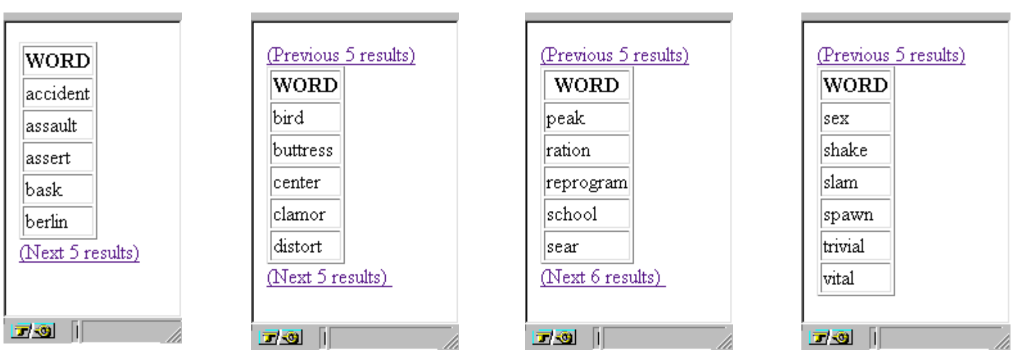
The second *dtml-in* tag simply displays the rows in the batch. The third *dtml-in* tag is similar to the first *dtml-in* tag, except that a hyperlink to the next batch, rather than the previous batch, is displayed. Table 24 shows the query string, previous batch URL and next-batch URL for the example shown in figure 7.

Orphan rows

Note that in the previous example the size of the last batch is six. This is because the *dtml-in* tag has a feature which attempts to prevent the display of very small batches by combining them with adjacent batches. Normally, if the number of rows in a batch is less than or equal to two, then it is combined with an adjacent batch. The *orphan* attribute in the *dtml-in* tag can be used to provide an alternative setting. The value provided in an *orphan* attribute is the desired minimum batch size. The default setting for an *orphan* attribute is 3 (three).

```
<dtml-var standard_html_header>

<dtml-if w36>

  <dtml-in w36 previous size=5 start=qs>
    <a href="<dtml-var document_id><dtml-var sequence-query
           >qs=<dtml-var previous-sequence-start-number>">
    (Previous <dtml-var previous-sequence-size> results)</a>
  </dtml-in>

  <table border>
    <tr><th>WORD</th></tr>
    <dtml-in w36 size=5 start=qs>
      <tr><td><dtml-var WORD></td></tr>
    </dtml-in>
  </table>

  <dtml-in w36 next size=5 start=qs>
    <a href="<dtml-var document_id><dtml-var sequence-query
             >qs=<dtml-var next-sequence-start-number>">
    (Next <dtml-var next-sequence-size> results)</a>
  </dtml-in>

<dtml-else>
  Sorry, no words.
</dtml-if>

<dtml-var standard_html_footer>
```

Figure 6. DTML source to browse 36 words, 5 words at a time



(words 1-5) (words 6-10) (words 26-30) (words 31-36)

Figure 7. The output of the DTML source in figure 6 as displayed in a Web browser for several batches.

Words	Query string	Previous-batch URL	Next- batch URL
1-5			r36?qs=6
6-10	qs=6	r36?qs=1	r36?qs=11
26-30	qs=26	r36?qs=21	r36?qs=31
31-36	qs=31	r36?qs=26	

TABLE 24. Query strings and previous batch URL[i] and next batch URL for the batches shown in figure 7

i. These are relative URL's as generated by the DTML. The document_id for this example is r36.

Overlapping batches

Normally, batches are non-overlapping. For large batch sizes, it is sometimes useful to overlap rows between batches. The *overlap* attribute in the *in* tag is used to specify how many rows to overlap between batches. The *overlap* attribute's default value is 0.

Showing row number and row data in previous and next batch hyperlinks.

The variables beginning *previous-sequence-start-*, *previous-sequence-end-*, *next-sequence-start-* and *next-sequence-end-* and ending in *index*, *number*, *roman*, *Roman*, *letter*, *Letter*, and *var-xxx*, where *xxx* is a row attribute (or key) name, can be used to label which rows begin and end previous and next batches. This is illustrated in figures 8 and 9, which use various batch insertion variables to label previous and next batches.

```
<dtml-in w36 previous size=5 start=qs>
  (<dtml-var previous-sequence-start-roman> -
   <dtml-var previous-sequence-end-roman>)
</dtml-in>

<table border><tr><th>WORD</th></tr>
  <dtml-in w36 size=5 start=qs>
    <tr><td><dtml-var WORD></td></tr>
  </dtml-in>
</table>

<dtml-in w36 next size=5 start=qs>
  (<dtml-var next-sequence-start-number> -
   <dtml-var next-sequence-end-number>)
</dtml-in>
```

Figure 8. Using batch-processing variables to number previous and next batch rows using Roman and Arabic numerals.

```
<dtml-in w36 previous size=5 start=qs>
  (<dtml-var previous-sequence-start-letter> -
      <dtml-var previous-sequence-end-letter>)
</dtml-in>

<table border><tr><th>WORD</th></tr>
  <dtml-in w36 size=5 start=qs>
    <tr><td><dtml-var WORD></td></tr>
  </dtml-in>
</table>

<dtml-in w36 next size=5 start=qs>
   (<dtml-var next-sequence-start-var-WORD> -
    <dtml-var next-sequence-end-var-WORD>)
</dtml-in>
```

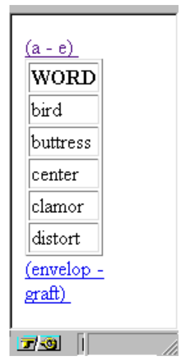


Figure 9. Using batch-processing variables to number previous-batch rows using letters and next-batch starting and ending words.

Showing information about multiple batches

Hyperlinks to multiple batches can be provided using the *next-batches* and *previous-batches* variables. These variables provide access to sequences of mapping objects containing information about all previous and next batches. Figure 10 implements the use of *previous-batches* to show hyperlinks to previous batches using starting and ending row numbers and *next-batches* to show hyperlinks to next batches using starting and ending words. Note that nested *dtml-in* tags are used to iterate over batch information. The nested *dtml-in* tags must use the *mapping* attribute because the items in the sequences, associated with *next-batches* and *previous-batches*, are mapping objects.

```
<dtml-in w36 previous size=5 start=qs>
  <dtml-in previous-batches mapping>
    <dtml-unless sequence-start>, </dtml-unless>
    <a href="<dtml-var document_id><dtml-var sequence-query
             >qs=<dtml-var batch-start-number>">
     <dtml-var batch-start-number> -
     <dtml-var batch-end-number></a>
  </dtml-in>
</dtml-in>

<table border><tr><th>WORD</th></tr>
  <dtml-in w36 size=5 start=qs>
    <tr><td><dtml-var WORD></td></tr>
  </dtml-in>
</table>

<dtml-in w36 next size=5 start=qs>
  <dtml-in next-batches mapping>
    </dtml-unless sequence-start>, <dtml-unless>
    <a href="<dtml-var document_id><dtml-var sequence-query
             >qs=<dtml-var batch-start-number>">
     <dtml-var batch-start-var-WORD> -
     <dtml-var batch-end-var-WORD></a>
  </dtml-in>
</dtml-in>
```

Figure 10. Use of DTML to provide links to all previous and next batches.

Displaying Objects with the *dtml-with* Tag

The *dtml-with* tag can be used to expand the namespace of a document template by adding attributes (or mapping keys) from an object which already exists in the document template namespace. For example, if a document template is used to display a folder, the contents of a sub-folder can be displayed using a *dtml-with* tag:

```
<dtml-with subfolder>
   <dtml-var title>
</dtml-with>
```

In combination dtml-with the *namespace* method of the special variable, _, the *dtml-with* tag can be used to add new variables to the DTML name space:

```
<dtml-with "_.namespace(profit=price-cost, title=product_name+’ summary’)">
  <h3><dtml-var title></h3>
  The profit is <dtml-var profit>
</dtml-with>
```

A common use of the *dtml-with* tag is to cause request variables to be used before object attributes:

```
The current id is <dtml-var id>
<dtml-with REQUEST>
   The id you entered was <dtml-var id>
</dtml-with>
```

Normally, document templates are used as methods of objects. Object attributes commonly take precedence over request variables. Using the *REQUEST* variable in a *dtml-with* tag causes the request to be searched before other parts of the name space.

Using the *only* Attribute to Limit the Namespace

The *only* attribute, unique to the *dtml-with* tag, prunes the enclosing namespaces when rendering the body of the tag. This is advantageous to prevent acquisition within DTML:

```
<dtml-with REQUEST only>
   <dtml-unless id>
      An id was not specified.
   <dtml-/unless>
<dtml-/with>
```

Without the *only* attribute, the above DTML would likely get an id value from the enclosing environment, which, in this example, is unwanted.

Multiple assignments with the *dtml-let* Tag

The *dtml-let* tag works like the *dtml-with* tag. It is more flexible in that it allows you to make multiple assignments; and allows you to chain assignments, using earlier declarations in later assignments.

The *dtml-let* tag is a new tag that lets you create blocks like:

```
<dtml-in "1,2,3,4">
<dtml-let num=sequence-item
    index=sequence-index
    result="num*index">
<dtml-var num> * <dtml-var index> = <dtml-var result>
</dtml-let>
</dtml-in>
```

Which yields:

```
1 * 0 = 0
2 * 1 = 2
3 * 2 = 6
4 * 3 = 12
```

Notice in the above example, the `'result'` variable is based on `'num'`, and `'index'`, both of which are assigned in the same *dtml-let* expression.

The syntax of the *dtml-let* tag requires that each argument to be evaluated in the head of the *dtml-let* tag must be separated by a newline. Enclosing an argument in double quotes causes it to be evaluated by the DTML expression machinery (*"num*index"*). Unquoted arguments are referenced by name.

Evaluation of the *dtml-let* tag is in sequence. The results of earlier assignments are available in later assignments. Later assignments can also override earlier ones, which can be helpful for longer step-by-step calculations. The variables set are in effect for the life of the *dtml-let* block.

Evaluating Names or Expressions without Generating Text Using the *dtml-call* Tag

Sometimes, document templates are used to perform actions in addition to or even instead of displaying results. Methods can be called when evaluating *name* attributes or in *expr* attribute expressions. These methods may perform a useful action but produce no output or produce an output which is not needed. The *dtml-call* tag is provided for evaluating expressions or calling methods that have a useful side effect without inserting any text:

```
<dtml-call "addDocument('hi','display a greeting','Hello world!')">
```

Reporting Errors with the *dtml-raise* Tag

In many applications, inputs or other variables need to be checked for validity before actions are performed. DTML provides a convenient means of performing validity checks by using the *dtml-raise* tag in combination with the *if* tag. Validity checks are performed with the *if* tag. The *dtml-raise* tag is used to report the errors.

The *dtml-raise* tag has a *type* attribute for specifying an error type. Like the standard *name* attribute, the attribute name of the *type* attribute may be omitted. The error type is a short descriptive name for the error. In addition, there are some standard types, like "*Unauthorized*" and "*Redirect*" that are returned as HTTP errors. In particular, "*Unauthorized*" errors cause a log-in prompt to be displayed on the user's browser.

The *dtml-raise* tag is a non-empty tag. The source enclosed by the *dtml-raise* tag is rendered to create an error message. If the rendered text contains any HTML markup, then Zope will display the text as an error message on the browser, otherwise a generic error message is displayed.

Here is a *dtml-raise* tag example:

```
<dtml-if "balance >= debit_amount">
  <dtml-call "debitAccount(account)">
  Your account, <dtml-var account>, has been debited.
<dtml-else>
  <dtml-raise type="Insufficient funds">
     There is not enough money in account <dtml-account>
     to cover the requested debit amount.<p>
  </dtml-raise>
</dtml-if>
```

The *dtml-raise* tag causes a Zope error to occur. However, there is an important side effect to this error that causes any changes made by a web request to be ignored, assuming that a transactional persistence mechanism, like the Zope Database is being used.

Exception Handling with the *dtml-try* Tag

Exceptions are unexpected errors Zope encounters during the rendering of a DTML statement. Once an exception is detected, the normal execution of the DTML stops. Consider the following example:

```
Cost per unit: $<dtml-var expr="_.float(total_cost/total_units)">
```

This statement functions normally if *total_units* is not zero. However, in the event that *total_units* is zero, a ZeroDivisionError exception is raised indicating an illegal operation. Thus, rather than rendering the DTML, an error message will be returned.

DTML provides the *dtml-try* tag to catch and handle these problematic exceptions within a block of DTML code. This allows you to anticipate and handle errors yourself, rather than getting a Zope error message whenever an exception occurs.

As an exception handler, the *dtml-try* tag has two functions. First, if an exception is raised, the *dtml-try* tag gains control of execution and handles the exception appropriately, and thus avoids returning a Zope error message. Second, the *dtml-try* tag allows the rendering of any subsequent DMTL to continue.

Within the *dtml-try* tag are one or more *dtml-except* tags that identify and handle different exceptions. When an exception is raised, each *dtml-except* tag is checked in turn to see if it matches the exception's type. The first *dtml-except* tag to match handles the exception. If no exceptions are given in a *dtml-except* tag, then the *dtml-except* tag will match all exceptions.

Implementing the *dtml-try* tag in the example above would resemble:

```
<dtml-try>
   Cost per unit: $<dtml-var expr="_.float(total_cost/total_units")>
   <dtml-except ZeroDivisionError>
       Cost per unit: N/A
</dtml-try>
```

If a ZeroDivisionError is raised, control goes to the *dtml-except* tag, and "Cost per unit: N/A" is rendered. Once the statements of the *dtml-except* tag finish, execution of DTML continues past the *dtml-try* block.

DTML's *except* tags work with Python's class-based exceptions. In addition to matching exceptions by name, the dtml-except tag will match any subclass of the named exception. For example, if ArithmaticError is named in a *dtml-except* tag, the tag can handle all ArithmaticError subclasses including, ZeroDivisionError.

Inside the body of a *dtml-except* tag you can access information about the handled exception through several special variables. These variables are:

1. *error_type*, which represents the type of the handled exception,
2. *error_value*, which represents the value of the handled exception, and
3. *error_tb*, which represents the traceback of the handled exception.

The dtml-try tag optional dtml-else block

The *dtml-try* tag has an optional *dtml-else* block that is rendered if an exception didn't occur. The exceptions in the else block are not handled by the preceding *dtml-except* blocks. Implementing the *dtml-else* tag with the *dtml-try* tag would be like:

```
<dtml-try>
    <dtml-except SomeError AnotherError>
    <dtml-except YetAnotherError>
    <dtml-except>
        <dtml-else>
</dtml-try>
```

The first *dtml-except* block to match the type of error raised is rendered. If a *dtml-except* block has no name, then it matches all raised errors. The optional *dtml-else* block is rendered when no exception occurs in the *dtml-try* block. Exception in the *dtml-else* block are not handled by the preceding *dtml-except* blocks.

The dtml-try tag optional dtml-finally block

In addition to the *dtml-else* block, the *dtml-try* tag has the ability to use a *dtml-finally* block that is always rendered whether an exception occurs or not. The *dtml-finally* form specifies a **cleanup** block to be rendered even when a exception occurs. Note, any rendered results are discarded if an exception occurs in either the *try* or *finally* blocks. The *dtml-finally* block is only of any used if you need to clean up something that will not be cleaned up by the transaction abort code. The *dtml-finally* block will always be called, whether there is an exception or nor and whether a return tage is used or not. If you use a *dtml-return* tag in the try block, any output of the *dtml-finally* block is discarded.

```
<dtml-try>
    <dtml-finally>
</dtml-try>
```

Important to note, if a exception occurs in the *dtml-try* block and an exception occurs in the *dtml-finally* block any information about the first exception is lost. It follows that if a *return tag* is used in the *dtml-try* block and an exception occurs in the *dtml-finally* block, the result returned in the *dtml-try* block will be lost. Also, if a *return tag* is used in the *dtml-finally* block, the result returned in the *dtml-try* block will be lost as well.

Excluding Source Text with the *dtml-comment* Tag

The *dtml-comment* tag provides a way to exclude source text from the rendered text. The *dtml-comment* tag is a simple non-empty tag that inserts no text. Further, the source enclosed by a *dtml-comment* tag is not rendered. *Dtml-comment* tags can be used to provide explanatory text or to disable parts of the source. *Comment* tags can be nested. Here is an example:

```
<dtml-call updateData>
    The data have been updated.
<dtml-comment>
   This comment is used to disable logging.

   <dtml-comment>
       The following call records that updates were made
   </dtml-comment>

   <dtml-call logUpdates>
</dtml-comment>
```

Returning Data using the *dtml-return* tag

The *dtml-return* tag is used to return data rather than text from a DTML method. It provides a way to use DTML methods to perform simple computation that can be used by other DTML methods and Zope objects. The only attributes supported by the *dtml-return* tag are the standard *name* and *expr* attributes.

Consider the following example:

```
blah blah
<dtml-return "1">
```

When this DTL method is executed, the number 1 is returned. The text "*blah blah*" is not returned. In:

```
<dtml-in objectIds>
   <dtml-return sequence-item>
</dtml-in>
blah
```

If there are any object ids, then the first one is returned, otherwise, the DTML text, "*blah*" is returned.

Displaying Information Hierarchically: the *dtml-tree* tag

Zope objects are organized and presented in a hierarchical fashion, so it is only natural that there should be a DTML facility to aid in the hierarchical display of information. This facility is the *dtml-tree* tag. The *dtml-tree* tag is similar to the *dtml-in* tag, in that it is applied to objects that contain other objects. However, in addition to iterating over sub-objects, the *dtml-tree* tag provides the ability to expand and iterate over sub-objects of sub-objects.

The *dtml-tree* tag provides many options which allow you quite a bit of control over how the tree is displayed.

Figure 11 shows two tree views of a Zope site with and without a folder expanded. This view is generated with the DTML shown in figure 12.

(a) (b)

Figure 11. A tree view of a folders in a Zope “school” site with
(a) no top-level folders expanded and (b) the Grades folder expanded

```
<dtml-var standard_html_header>

<dtml-tree>
  <dtml-var id>
</dtml-tree>

<dtml-var standard_html_footer>
```

Figure 12. DTML source to produce the tree views shown in figure 11.

This example shows an extremely simple form of the *dtml-tree* tag. No attributes are used. Although the *dtml-tree* tag can use the standard *name* and *expr* attributes to specify data to be displayed, these attributes may be and usually are omitted. The *dtml-tree* tag usually uses the current folder as the source of data to be displayed. The text following the *dtml-tree* tag is inserted for each “branch” of a tree. The attributes that can be used with the *dtml-tree* tag are summarized in table 21.

By default, the *dtml-tree* tag displays branches and sub-branches of an object. Branches are normally found by calling a method named *tpValues* of the object being displayed by the dtml-tree tag. Many Zope objects, including folders, provide *tpValues* methods. Alternatively, the branches method may be used to specify a different method to call to find branches. For example, to display all sub-objects of a folder, the *objectValues* method may be used (figure 13).

Name	Value required?	Description
name	yes	Insert the name of the variable to be inserted.
expr	yes	Insert an expression that evaluates to the value to be inserted.
branches	yes	The name of the method used to find sub-objects. This defaults to tpValues, which is a method defined by a number of objects in Zope and in Zope products.
branches_expr	yes	An expression which is evaluated to find sub-objects. This attribute performs the same function as the branches attribute but uses an expression rather than the name of a method.
id	yes	The name of a method or attribute used to determine the id of an object for the purposes of calculating tree state. This defaults to tpId which is a method defined by many Zope objects. This attribute is mostly useful for developers who wish to have fine control of the internal representation of the tree state.
url	yes	The name of a method or attribute used to determine the url of an object. This defaults to tpURL which is a method defined by many Zope objects. This attribute is mostly useful for developers who wish to have fine control over tree url generation.
leaves	yes	The name of a Document used to expand sub-objects that do not have sub-object branches.
header	yes	The name of a Document to be shown at the top of each expansion. This provides an opportunity to “brand” a branch in a hierarchy. If the named document cannot be found for a branch, then the header attribute is ignored for that branch.
footer	yes	The name of a Document to be shown at the bottom of each expansion. If the named document cannot be found for a branch, then the footer attribute is ignored for that branch.
nowrap	yes	Either 0 or 1. If 0, then branch text will wrap to fit in available space, otherwise, text may be truncated. The default value is 0.
sort	yes	Sort branches before text insertion is performed. The attribute value is the name of the attribute that items should be sorted on.
assume_children	yes	Either 0 or 1. If 1, then all items are assume to have sub-items, and will therefore always have a plus sign in front of them when they are collapsed. Only when an item is expanded will sub-objects be looked for. This could be a good option when the retrieval of sub-objects is a costly process.
single	yes	Either 0 or 1. If 1, then only one branch of the tree can be expanded. Any expanded branches will collapse when a new branch is expanded.
skip_unauthorized	yes	Either 0 or 1. If 1, then no errors will be raised trying to display sub-objects to which the user does not have sufficient access.

TABLE 25. *dtml-tree* tag attributes.

An object that does not have sub-branches may instead define “leaves” by using the *leaves* attribute of the *dtml-tree* tag. The argument to the *leaves* attribute is a *Document* object. This is commonly used when browsing database data. Branches are used to provide a hierarchical organization to data and leaves are used to display data within a hierarchical grouping.

The *header* and *footer* attributes are similar to the *leaves* attribute, in that they are used to specify documents to be displayed when a branch is expanded. Unlike the *leaves* attribute, they are only used when there are sub-branches of an object. The *header* document is displayed before the display of sub-branches, and the *footer* is displayed following sub-branches.

```
<dtml-var standard_html_header>

<dtml-tree branches=objectValues>
  <IMG SRC="<dtml-var SCRIPT_NAME
       >/<dtml-var icon>">
  <dtml-var id>
</dtml-tree>

<dtml-var standard_html_footer>
```

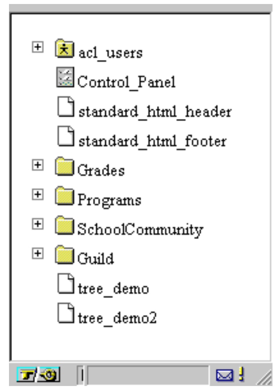


Figure 13. Use of the *branches* tag to display all sub-objects in a folder.

The *dtml-tree* tag sets a number of variables in the DTML namespace as it renders sub-objects. These variables allow sub-objects to tailor their representation to their position within the tree. Perhaps the most useful variable set by the *dtml-tree* tag is the *tree-item-expanded* variable. If this variable is true then the tree item knows that it has been expanded. The variables set by the tree tag are summarized in Table 26.

Name	Description
tree-item-expanded	True is the current item is expanded.
tree-item-url	The URL of the current item relative to the URL of the DTML document in which the tree tag appears. This variable relies on the tree tag's url attribute to generate the tree-item URL
tree-root-url	The URL of the DTML document in which the tree tag appears.
tree-level	The depth of the current item. Items at the top of the tree have a level of 0.
tree-colspan	The number of levels deep the tree is being rendered. This variable along with the tree-level variable can be used to calculate table rows and colspan settings when inserting table rows into the tree table.
tree-state	The tree state expressed as a list of ids and sub-lists of ids. This variable is mostly of interest to developers who which to have precise knowledge of a tree's state.

TABLE 26. Variables set by the dtml-tree tag when rendering sub-objects.

Additionally, the *dtml-tree* tag responds to several variables set the DTML namespace. You can expand and collapse the entire tree by setting the *expand_all* and *collapse_all* variables. Table 27 details the variables which control tree state.

One common application of these variables is to provide links which allow a tree to be expanded and collapsed. Here's an example of how this could be done in DTML:

```
<P>
<a href="<dtml-var URL0>?expand_all=1">Expand tree</a> |
<a href="<dtml-var URL0>?collapse_all=1">Collapse tree</a>
</P>
```

This snippet of DTML provides two links to the current page. One link will cause the current page's tree to expand, the other will cause it to collapse.

Name	Description
expand_all	If set to a true value, this variable causes the entire tree to be expanded.
collapse_all	If set to a true value, this variable causes the entire tree to be collapsed.
tree-s	This variable contains the tree state in a compressed and encoded form. This variable is set in a cookie to allow the tree to maintain its state. Developers may control the state of the tree directly by setting this variable, though this is not recommended.
tree-e	This variable contains a compressed and encoded list of ids to expand. Developers may control the state of the tree directly by setting this variable, though this is not recommended.
tree-c	This variable contains a compressed and encoded list of ids to collapse. Developers may control the state of the tree directly by setting this variable, though this is not recommended.

TABLE 27. Variable that influence the dtml-tree tag

Sending Mail: the *dtml-sendmail* tag

The *dtml-sendmail* tag is used to send an electronic message using the Simple Mail Transport Protocol (SMTP). Unlike other DTML tags, the *dtml-sendmail* tag does not cause any text to be included in output. Figure 14 shows a DTML document, named `SendFeedback`, which uses the *dtml-sendmail* tag to send information collected in a feedback form (figure 15). The *dtml-sendmail* tag requires numerous pieces of information that are specified by *dtml-sendmail* tag attributes, shown in Table 28. At minimum, either a Zope `MailHost` object must be specified, or a SMTP host address must be specified using a *smtphost* attribute. The recipients, sender, and subject information are required, but they may be provided either as *dtml-sendmail* tag attributes, or using "header" lines[1] at the beginning of the message (figure 14).

```
<dtml-var standard_html_header>

<dtml-sendmail smtphost="gator.digicool.com">
To: Product Support <<dtml-var support>>
From: Web Feedback Form <<dtml-var feedback>>
Subject: <dtml-var subject>

<dtml-var body>
</dtml-sendmail>

Thank you for your input!

<dtml-var standard_html_footer>
```

Figure 14. A sample document that uses the *dtml-sendmail* tag

```
<dtml-var standard_html_header>

<H2>We want your input!</H2>

<form action=SendFeedback>
  Subject: <input type=text name=subject size=40>
 <br>
  <textarea name=body rows=10 cols=50>
  </textarea><br>
  <input type=submit value="Send Feedback">
</form>

<dtml-var standard_html_footer>
```

Figure 15. A feedback form that collects a subject and body from a user and submits them to the SendFeedback document shown in figure 14.

1. A header line is a line that begins with a header name, followed by a colon, a space, and a value. Header lines may have continuation lines that begin with one or more spaces or tabs. All of the header lines, with continuation lines, if any, must start at the beginning of the text following the dtml-sendmail tag and must be separated from the message body by at least one blank line.

Name	Description
mailhost	A Zope MailHost object that manages Simple Mail Transport Protocol (SMTP) and port information. This attribute is not used if the smtphost attribute is used.
smtphost	The address of a SMTP server. Mail will be delivered to this server, which will do most of the work of sending mail. This attribute is not used if the mailhost attribute is used.
port	If the smtphost attribute is used, then the *port* attribute is used to specify a port number to connect to. If not specified, then port 25 will be used.
mailto	A recipient address or a list of recipient addresses separated by commas.
mailfrom	A sender address.
subject	The subject of the message.

TABLE 28. *dtml-sendmail* tag attributes

The text following the *dtml-sendmail* tag can and usually does use DTML tags to include data from input forms and Zope objects. In the `SendFeedback` example in figure 14, the variables `support` and `feedback` are supplied from Folder properties and the variables `subject` and `body` are supplied from the `FeedbackForm` *Document* shown in figure 15.

Sending Attachments with the *dtml-mime* Tag

The *dtml-mime* tag is used in conjunction with the *dtml-sendmail* tag to send attachments along with electronic mail messages[1]. The *dtml-mime* tag automatically sets the content type of the entire message to multipart/mixed. Thus, a variety of data can be attached to a single message using one or more *dtml-boundary* tags. Figure 16 uses the *dtml-mime* tag to attach the file, yearly_report, to an email formed by the *dtml-sendmail* tag.

```
<dtml-var standard_hmtl_header>
<dtml-sendmail smtphost=gator.digicool.com>
From: zope@digicool.com
To: <dtml-var who>
<dtml-mime type=text/plain encode=7bit>

Here is the yearly report.

<dtml-boundary type=application/octet-stream
disposition=attachment encode=base64><dtml-var
"yearly_report">
</dtml-mime>

</dtml-sendmail>

Mail with attachment was sent.
<dtml-var standard_hmtl_footer>
```

Figure 16. The *dtml-mime* tag used to attach a file to an email message

The *dtml-mime* tag and *dtml-boundary* tags contain several attributes, listed in Table 19, that specify MIME header information for their particular content. Since the opening *dtml-mime* tag in Figure 16 contains the body of the message, and does not require encoding, *encode* is set to 7bit.

Notice in Figure 16, there is no space between the opening *dtml-mime* tag and the *TO:* header. If a space is present, then the message will not be interpreted as by the receiving mailreader. Also notice, there are no spaces between the *dtml-boundary*, *var* or closing *dtml-mime* tags.   I

Name	Description
type	Sets the MIME header, Content-Type, of the subsequent data.
disposition	Sets the MIME header, Content-Disposition, of the subsequent data. If disposition is not specified in a mime or boundary tag, then Content-Disposition MIME header is not included
encode	Sets the MIME header, Content-Transfer-Encoding, of the subsequent data. If encode is not specified, base64 is used as default. The options for encode are: base64, uuencode, x-uuencode, quoted-printable, uue, x-uue, and 7bit. No encoding is done if set to 7bit.

TABLE 29. *dtml-mime* tag attributes

1. The *dtml-mime* tag can also be used in http multipart responses.

Appendix A, Date-time data

Zope provides a facility for working with DateTime data. From Python, a DateTime module provides a DateTime class for creating and formatting DateTime data. From DTML, the special variable _ provides a method for constructing date-time values from strings and numeric data. DateTime objects provide methods that can be used to format data in various ways.

Name	Description
AMPM	Return the time string for an object to the nearest second.
AMPMMinutes	Return the time string for an object not showing seconds.
aCommon	Return a string representing the object's value with the format: Mar 1, 1997 1:45 pm.
aCommonZ	Return a string representing the object's value with the format: Mar 1, 1997 1:45 pm US/Eastern.
aDay	Return the abbreviated name of the day of the week.
aMonth	Return the abbreviated month name.
ampm	Return the appropriate time modifier (am or pm).
Date	Return the date string for the object.
Day	Return the full name of the day of the week.
DayOfWeek	Compatibility: see Day.
day	Return the integer day.
dayOfYear	Return the day of the year, in context of the time-zone representation of the object.
dd	Return day as a 2 digit string.
fCommon	Return a string representing the object's value with the format: March 1, 1997 1:45 pm.
fCommonZ	Return a string representing the object's value with the format: March 1, 1997 1:45 pm US/Eastern.
HTML4	The W3C recommended date & time format YYYY-MM-DDTHH:MM:SSZ. T,Z are literal characters. Time is in UTC.
h_12	Return the 12-hour clock representation of the hour.
h_24	Return the 24-hour clock representation of the hour.
hour	Return the 24-hour clock representation of the hour.
isCurrentHour	Return true if this object represents a date/time that falls within the current hour, in the context of this object's time-zone representation.
isCurrentMonth	Return true if this object represents a date/time that falls within the current month, in the context of this object's time-zone representation.
isFuture	Return true if this object represents a date/time later than the time of the call.
isLeapYear	Return true if the current year (in the context of the object's time zone) is a leap year.
isPast	Return true if this object represents a date/time earlier than the time of the call.
*Mon_*	Return the full month name.
Mon	Compatibility: see aMonth.
Month	Return the full month name.

TABLE 30. Custom formats for date-time data

Name	Description
minute	Return the minute.
mm	Return month as a 2 digit string.
month	Return the month of the object as an integer.
notEqualTo(t)	Compare this DateTime object to another DateTime object OR a floating point number, such as that which is returned by the python time module. Returns true if the object represents a date/time not equal to the specified DateTime or time module style time.
PreciseAMPM	Return the time string for the object.
PreciseTime	Return the time string for the object.
pCommon	Return a string representing the object's value with the format: Mar. 1, 1997 1:45 pm.
pCommonZ	Return a string representing the object's value with the format: Mar. 1, 1997 1:45 pm US/Eastern.
pDay	Return the abbreviated (with period) name of the day of the week.
pMonth	Return the abbreviated (with period) month name.
rfc822	Return the date in RFC 822 format.
second	Return the second.
TimeMinutes	Return the time string for an object not showing seconds.
Time	Return the time string for an object to the nearest second.
timezone	Return the time zone in which the object is represented.
year	Return the calendar year of the object
yy	Return calendar year as a 2 digit string.

TABLE 30. Custom formats for date-time data

INDEX

Symbols
_variable, 16

A
abs, 9
aCommon, 57
aCommonZ, 57
acos, 10
aDay, 57
Alternate Python String Format Syntax, 5
aMonth, 57
AMPM, 57
ampm, 57
AMPMMinutes, 57
appendHeader, 17
asin, 10
assume_children, 51
atan, 10
atan2, 10
atof, 12
atoi, 12
attributes, 3
AUTH_TYPE, 18
AUTHENTICATED_USER, 16, 18
AUTHENTICATION_PATH, 16

B
BASEn request variables, 15
batch
  end-index, 34
  start-index, 34
betavariate, 11
Bobo, 1, 15, 16, 18, 21, 22
boundary, 56
branches, 51
branches_expr, 51

C
capitalize, 4, 12, 23
capwords, 12
ceil, 10
CGI-defined request variables, 17
choice, 13
choice method of the whrandom attribute of the _ special variable used in expr attribute expressions, 11
chr, 9
collection-length, 24
comment, 17
CONTENT_LENGTH, 18
CONTENT_TYPE, 18
cookies, 16
cos, 10
cosh, 10
count, 12
count-nnn, 33
Creating document templates, 21
cunifvariate, 11

D
Date, 57
Day, 57
day, 57
DayOfWeek, 57
dayOfYear, 57
dd, 57
digits, 12
divmod, 9
Document template classes, 21
document templates, 15
document_id, 15
document_title, 15
dollars-and-cents, 24
domain, 17
dtml-if, 27

E
e attribute, 10
electronic mail messages, 56
elif tag, 27
else tag, 27
encode, 56
end tags, 4
error_tb, 46
error_type, 46
error_value, 46
etc, 23, 25
except, 46
except tag, 46
exceptions, 46
exp, 10, 51
expireCookie, 17
expires, 17
expovariate, 11
expr, 15, 16, 23, 30, 50
expr attribute, 4
expr attribute, variable lookup, 8
extended Python string format syntax, 5

F
fabs, 10
fCommon, 57
fCommonZ, 57
find, 12
float, 9
floor, 10
fmod, 10
fmt, 23
footer, 51
form, 16
frexp, 10

G
gamma, 11
GATEWAY_INTERFACE, 17
gauss, 11
getattr, 9
getHeader, 17
getitem, 9
GetStatus, 17

H
h_12, 57
h_24, 57
has_key, 16
hasattr, 9
hash, 9
header, 51
hex, 9
hexdigits, 12
hour, 57
html_quote, 23
HTML4, 57
HTTP headers, 15
HTTP_REFERER, 15
hypot - hypotenuse, 10

I
id, 51
if, 45
if tag, 4
index, 12
insertion-by-name, 5
instance attributes, 18
int, 9
intermediate tags, 4
isCurrentHour, 57
isCurrentMonth, 57
isFuture, 57
isLeapYear, 57
isPast, 57
item, 16

J
join list or tuple, 13
Justification
  center, 13
  left, 13
  right, 13

K
keys, 16
key-value pairs, 16

L
ldexp, 10
leaves, 51
len, 9
let tag, 42
letters, 12
log, 10
log10, 10
lognormvariate, 11
lower, 13, 23
lowercase, 12

M
maketrans, 13
mapping, 30, 39
math, 9
max, 9
max_age, 17
max-nnn, 33
mean-nnn, 33
median-nnn, 33
mime tag
  disposition, 56
  encode, 56
  type, 56
min, 9
min-nnn, 33
minute, 58
missing, 23
mm, 58
modf, 10
Mon, 57
Mon_, 57
Month, 57
month, 58

N
name, 23, 30, 50, 51
name attribute, 3
namespace, 9, 41
newline_to_br, 23
next, 30
next attribute of the in tag, 30
next-batches, 39
next-sequence, 34
  end, 37
  end-index, 34
  next-batches, 34
  size, 34
  start, 37
  start-index, 34
None, 9
normalvariate, 11
notEqualTo(t), 58
nowrap, 51
null attribute, 23

O
objectValues, 50
oct, 9
octdigits, 12
ord, 9
orphan, 30
overlap, 30, 37

P
PARENTS, 16
paretovariate, 11
path, 17
PATH_INFO, 18
PATH_TRANSLATED, 18
pCommon, 58
pCommonZ, 58
pDay, 58
pi, 10
pMonth, 58
pow, 9
power, 10
PreciseAMPM, 58
PreciseTime, 58
previous, 30
previous-batches, 39
previous-sequence, 34
  batches, 34
  end, 37
  end-index, 34
  previous-batches, 34
  size, 34, 35
  start, 37
  start-index, 34
  start-number, 35
previous-sequence-size batch processing variable, 34
Principia, 1, 3, 7

Q
QUERY_STRING, 18

R
randint, 13
randint method of the whrandom attribute of the _ special variable used in expr attribute expressions, 11
random, 13
random method of the whrandom attribute of the _ special variable used in expr attribute expressions, 11
Redirect, 45
redirect, 17
REMOTE_ADDR, 18
REMOTE_HOST, 18
REMOTE_IDENT, 18
REMOTE_USER, 18
REQUEST, 16, 41
REQUEST_METHOD, 18
RESPONSE, 16
reverse, 30
rfc822, 58
rfind, 12
rindex, 12
round, 9

S
SCRIPT_NAME, 18
second, 58
secure, 17
seed, 13
sendmail, 56
  mailfrom, 55
  mailhost, 55
  mailto, 55
  port, 55
  smtphost, 55

subject, 55
sequence-end, 32
sequence-index, 32
sequence-item, 32
sequence-key, 32
sequence-query, 34, 35
sequence-start, 32
sequence-step-size, 34
SERVER_NAME, 17
SERVER_PORT, 17
SERVER_PROTOCOL, 17
SERVER_SOFTWARE, 17
server-side-include syntax, 3
set, 16
SetBase, 17
setCookie, 17
SetHeader, 17
setStatus, 17
shorthand form of the expr attribute, 4
shorthand version of the name attribute, 3, 4
Simple Mail Transport Protocol, 54
sin, 10
single, 51
sinh, 10
size, 30
size attribute, 23, 25, 34
size attribute of the in tag, 30
skip_unauthorized, 30, 51
smtphost, 54
sort, 29, 30, 51
spacify attribute, 23
split string, 13
sql_quote, 23
sqrt, 10
standard-deviation-nnn, 33
standard-deviation-n-nnn, 33
start, 30
start attribute, 34
str, 9
string, 9
strip, 13
leading white space, 13
trailing white space, 13
structured-text, 24
subclassing, 18
swapcase, 13

T

tag, 3
tan, 10
tanh, 10
thousands_commas, 23
Time, 58
TimeMinutes, 58
timezone, 58
total-nnn, 33
tpValues, 50
translate, 13
tree
collapse-all, 53
colspan, 52
expand-all, 53
level, 52
root-url, 52
state, 52
tree-c, 53
tree-e, 53
tree-s, 53
tree-item
expanded, 52
url, 52
tree-item-expanded, 52
try tag, 46
type, 45

U

Unauthorized, 45
uniform, 13
uniform method of the whrandom attribute of the _ special variable used in expr attribute expressions, 11
upper, 13, 23
uppercase, 12
URL, 16
url, 51
url_quote, 23
url_quote_plus, 23
URLn request variables, 15
Using document templates, 22
Using Document Templates from Python, 21
Using document templates with ZPublisher, 22

V

validate, 18
values, 16
var, 3, 4, 15, 56
var tag, 3
variance-nnn, 33
variance-n-nnn, 33
vonmisesvariate, 11

W

Web-request variables, 16
weibullvariate, 11
whitespace, 12
whole-dollars, 24
whrandom, 9
whrandom attribute of the _ special variable used in expr attribute expressions, 9

Y

year, 58
yy, 58

Z

zfill, 13
Zope, 45
ZopeTime, 15