

Interchange Tags Reference

Table of Contents

<u>1. Interchange Tag Reference.....</u>	<u>1</u>
<u>1.1. Tag Syntax.....</u>	<u>1</u>
<u>1.2. Looping tags and Sub-tags.....</u>	<u>5</u>
<u>2. Tags.....</u>	<u>13</u>
<u>2.1. accessories.....</u>	<u>13</u>
<u>2.2. and.....</u>	<u>21</u>
<u>2.3. area.....</u>	<u>22</u>
<u>2.4. attr_list.....</u>	<u>25</u>
<u>2.5. banner.....</u>	<u>26</u>
<u>2.6. bounce.....</u>	<u>27</u>
<u>2.7. calc.....</u>	<u>28</u>
<u>2.8. cart.....</u>	<u>29</u>
<u>2.9. catch.....</u>	<u>30</u>
<u>2.10. cgi.....</u>	<u>30</u>
<u>2.11. checked.....</u>	<u>31</u>
<u>2.12. control.....</u>	<u>32</u>
<u>2.13. control_set.....</u>	<u>33</u>
<u>2.14. counter.....</u>	<u>33</u>
<u>2.15. currency.....</u>	<u>35</u>
<u>2.16. data.....</u>	<u>36</u>
<u>2.17. default.....</u>	<u>37</u>
<u>2.18. description.....</u>	<u>38</u>
<u>2.19. discount.....</u>	<u>39</u>
<u>2.20. discount_subtotal.....</u>	<u>41</u>
<u>2.21. dump.....</u>	<u>41</u>
<u>2.22. ecml.....</u>	<u>41</u>
<u>2.23. either.....</u>	<u>42</u>
<u>2.24. error.....</u>	<u>43</u>
<u>2.25. export.....</u>	<u>45</u>
<u>2.26. field.....</u>	<u>46</u>
<u>2.27. file.....</u>	<u>46</u>
<u>2.28. filter.....</u>	<u>47</u>
<u>2.29. flag.....</u>	<u>52</u>
<u>2.30. fly_list.....</u>	<u>53</u>
<u>2.31. fly_tax.....</u>	<u>54</u>
<u>2.32. goto.....</u>	<u>55</u>
<u>2.33. handling.....</u>	<u>55</u>
<u>2.34. harness.....</u>	<u>56</u>
<u>2.35. href.....</u>	<u>57</u>
<u>2.36. html_table.....</u>	<u>57</u>
<u>2.37. if.....</u>	<u>57</u>
<u>2.38. import.....</u>	<u>63</u>
<u>2.39. include.....</u>	<u>65</u>
<u>2.40. index.....</u>	<u>65</u>
<u>2.41. input_filter.....</u>	<u>66</u>
<u>2.42. item_list.....</u>	<u>67</u>

Table of Contents

2.43. label	70
2.44. log	71
2.45. loop	71
2.46. mail	75
2.47. mvasp	76
2.48. nitems	76
2.49. onfly	77
2.50. options	78
2.51. or	78
2.52. order	79
2.53. page	80
2.54. perl	85
2.55. price	87
2.56. process	89
2.57. profile	90
2.58. query	90
2.59. read_cookie	91
2.60. record	92
2.61. region	93
2.62. row	93
2.63. salestax	95
2.64. scratch	95
2.65. scratchd	96
2.66. search_list	97
2.67. search_region	97
2.68. selected	98
2.69. set	99
2.70. set_cookie	100
2.71. seti	100
2.72. setlocale	101
2.73. shipping	102
2.74. shipping_desc	102
2.75. soap	103
2.76. sql	104
2.77. strip	105
2.78. subtotal	105
2.79. tag	106
2.80. time	109
2.81. timed_build	111
2.82. tmp	113
2.83. total_cost	114
2.84. tree	114
2.85. try	119
2.86. update	119
2.87. userdb	120
2.88. value	122
2.89. value_extended	122

Table of Contents

<u>3. User-defined Tags.....</u>	<u>125</u>
<u>3.1. Alias.....</u>	<u>126</u>
<u>3.2. attrAlias.....</u>	<u>126</u>
<u>3.3. CanNest.....</u>	<u>126</u>
<u>3.4. HasEndTag.....</u>	<u>127</u>
<u>3.5. Implicit.....</u>	<u>127</u>
<u>3.6. InsertHTML.....</u>	<u>127</u>
<u>3.7. InsideHTML.....</u>	<u>127</u>
<u>3.8. Interpolate.....</u>	<u>128</u>
<u>3.9. InvalidateCache.....</u>	<u>128</u>
<u>3.10. Order.....</u>	<u>128</u>
<u>3.11. PosRoutine.....</u>	<u>128</u>
<u>3.12. ReplaceAttr.....</u>	<u>129</u>
<u>3.13. ReplaceHTML.....</u>	<u>129</u>
<u>3.14. Routine.....</u>	<u>129</u>
<u>4. Standard Usertags.....</u>	<u>131</u>
<u>4.1. bar button.....</u>	<u>131</u>
<u>4.2. delete cart.....</u>	<u>131</u>
<u>4.3. fedex query.....</u>	<u>131</u>
<u>4.4. loc.....</u>	<u>131</u>
<u>4.5. reconfig wait.....</u>	<u>131</u>
<u>4.6. title bar.....</u>	<u>131</u>
<u>4.7. xml generator.....</u>	<u>131</u>
<u>4.8. button.....</u>	<u>131</u>
<u>4.9. email.....</u>	<u>131</u>
<u>4.10. formel.....</u>	<u>131</u>
<u>4.11. rand.....</u>	<u>131</u>
<u>4.12. save cart.....</u>	<u>131</u>
<u>4.13. ups query.....</u>	<u>131</u>
<u>4.14. convert date.....</u>	<u>131</u>
<u>4.15. email raw.....</u>	<u>131</u>
<u>4.16. get url.....</u>	<u>131</u>
<u>4.17. reconfig.....</u>	<u>131</u>
<u>4.18. summary.....</u>	<u>132</u>
<u>4.19. usertrack.....</u>	<u>132</u>
<u>4.20. db date.....</u>	<u>132</u>
<u>4.21. fcounter.....</u>	<u>132</u>
<u>4.22. load cart.....</u>	<u>132</u>
<u>4.23. reconfig time.....</u>	<u>132</u>
<u>4.24. table organize.....</u>	<u>132</u>
<u>4.25. var.....</u>	<u>132</u>
<u>A. Tag Entry Format.....</u>	<u>133</u>
<u>4.26. dummy.....</u>	<u>133</u>
<u>B. Template Parsing Order.....</u>	<u>137</u>

1. Interchange Tag Reference

Interchange functions are accessed via ITL, the Interchange Tag Language. The pages in a catalog may be mostly HTML, but they will use ITL tags to provide access to Interchange's functions. ITL is a superset of MML, or Minivend Markup Language. Minivend was the predecessor to Interchange.

These tags perform various display and modification operations for the user session. There nearly 100 standard predefined tags, and the `UserTag` facility allows you to create tags fully as powerful that perform your own functions.

1.1. Tag Syntax

ITL tags are similar to HTML in syntax, in that they accept parameters or attributes and that there are both *standalone* and *container* tags.

We will call an attribute a *parameter* if it may be called positionally or if it must be included (see the [parameter](#) and [attribute](#) sections below).

A standalone tag has no ending element, e.g.:

```
[value\_name]
```

This tag will insert the user's name, providing they have given it to you in a form. A container tag has both a beginning and an ending element, as in:

```
[if value name]  
You have a name.  It is [value name].  
[/if]
```

We will usually call the ITL tags **Interchange tags** or just **tags**.

1.1.1. Standard Syntax

The most common syntax is to enclose the tag with its parameters and attributes in `[square brackets]`. If the tag has an end tag, the tag and its end tag will delimit contained body text:

```
[tagname parameters attributes]Contained Body Text[/tagname]
```

Caveat -- macros: Some macros look like tags or end tags. For example, `[/page]` is a macro for ``. This allows you to conveniently write `[page href]Target[/page]`, but 'Target' is not treated as contained body text.

When using the `[tagname ...]` syntax, there must be no whitespace between the left bracket ('[') and the tagname.

If a tag name includes an underscore or dash, as in [item_list](#), a dash is just as appropriate (i.e. `item-list`). The two forms are interchangeable, except that an ending tag must match the tag (i.e., don't use `[item-list] list [/item_list]`).

1.1.1.1. HTML–Comment

ITL also allows you to use '<!--[' and ']'-->' as interchangeable alternatives to plain square brackets: [tagname] and <!--[tagname]--> are equivalent.

This allows you make your raw tags appear as comments to HTML browsers or editors. You might want to do this if your editor has trouble with ITL tags when editing Interchange page templates, or alternatively, if you want to use one .html file both as an Interchange template and as a static page delivered directly by your web server, without Interchange processing.

To properly HTML–comment contained body text, place your comment–style brackets appropriately, for example:

```
<!--[tagname] Contained Body Text [/tagname]-->
```

Note that you must include whitespace between the HTML comment delimiters and the square brackets if you wish to actually comment out tag output in the browser. For example, if [value name] expands to 'Bill':

```
<!--[value name]--> becomes Bill
<!-- [value name] --> becomes <!-- Bill -->
```

1.1.1.1.1. Technical notes

While '<!--[' and '[' are interchangeable, the Interchange parser does not replace ']'-->' with ']' unless it sees '<!--[' elsewhere in the page.

See the [Template Parsing Order](#) appendix if you are modifying the special administrative interface pages and intend to use this syntax.

1.1.2. HTML–Embedded

As an alternative syntax, you can usually embed an Interchange tag as an attribute within an HTML tag. This allows some HTML editors to work more easily with Interchange templates (though you should consider the above HTML–comment–style brackets first). The following is a basic example of HTML–Embedded syntax:

```
<HTMLtag MV="tagname positional parameters" other HTML attributes>
```

The following examples will each loop over any items in the shopping cart, displaying their part number, description, and price, but only IF there are items in the cart.

HTML syntax:

```
<TABLE MV="if items">
<TR MV="item-list">
<TD> [item-code] </TD>
<TD> [item-description] </TD>
<TD> [item-price] </TD>
</TR></TABLE>
```

Standard syntax:

```
[if items]
```



```
<TABLE>
[ item-list ]
<TR>
<TD> [item-code] </TD>
<TD> [item-description] </TD>
<TD> [item-price] </TD>
</TR>
[/item-list]</TABLE>
[/if]
```

Note -- Disabling HTML-embedded tags for performance:

Avoid the HTML-embedded usage if you can.

The foundation catalog included with interchange disables parsing of the HTML-embedded syntax. This is for better performance, since it saves the server from checking every HTML tag for interchange tag calls. This is done in the catalog by setting the pragma [no_html_parse](#) in `catalog.cfg`.

1.1.3. Named vs. Positional Parameters

There are two styles of supplying parameters to a tag — named and positional.

In the named style you supply a parameter/value pair just as most HTML tags use:

```
[ value name="foo" ]
```

The same thing can be accomplished for the [[value](#)] tag with

```
[value foo]
```

The parameter name is the first positional parameter for the [[value](#)] tag. Some people find positional usage simpler for common tags, and Interchange interprets them somewhat faster. If you wish to avoid ambiguity you can always use named calling.

In most cases, tags specified in the positional fashion will work the same as named parameters. The only time you will need to modify them is when there is some ambiguity as to which parameter is which (usually due to whitespace), or when you need to use the output of a tag as the parameter or attribute for another tag.

1.1.3.1. Interpolating parameters

If you wish to use the value of a tag within a parameter of another tag, you cannot use a positional call. You must also double-quote the argument containing the tag you wish to have expanded. For example, this will not work:

```
[ page scan se=[scratch somevar]]
```

To get the output of the [`scratch somevar`] interpreted, you must place it within a named and quoted attribute:

```
[ page href=scan arg="se=[scratch somevar]" ]
```

Note that the argument to **href** ('scan') did not need to be quoted; only the argument to **arg**, which contained a tag, needed the quotes.

1.1.4. Universal Attributes

Universal attributes apply to all tags, though each tag specifies its own default for the attribute. The code implementing universal attributes is external to the core routines that implement specific tags.

1.1.4.1. interpolate

This attribute behaves differently depending on whether the tag is a *container* or *standalone* tag. A container tag is one which has an end tag, i.e. `[tag] stuff [/tag]`. A standalone tag has no end tag, as in `[area href=somepage]`. (Note that `[page ...]` and `[order ..]` are **not** container tags.)

For container tags (interpolated)

- If true ("interpolate=1"), the interchange server will first process any tags within the body text before passing it to the enclosing tag.
- If false ("interpolate=0"), the enclosing tag will receive the raw body text.

For standalone tags (reparsed)

- If true, the server will process the *output* of the tag. This is identical to the behavior of the [reparse](#) attribute (see below for explanation and examples).

(Note: The mixing of 'interpolate' and 'reparse' logic occurred because 'interpolate' already worked this way when 'reparse' was added to interchange. This may be fixed in a later release...)

Most standalone tags are not reparsed by default (i.e., interpolate=0 by default). There are some exceptions, such as the [include](#) tag.

Interpolation example:

Assuming that name is 'Kilroy',

```
[log interpolate=1][value name] was here[/log]
[log interpolate=0][value name] was here[/log]
```

the first line logs 'Kilroy was here' to *catalog_root/etc/log*, while the second logs '[value name] was here'.

Reparsing example:

Suppose we set a scratch variable called 'now' as follows:

```
[set name=now interpolate=0][time]%A, %B %d, %Y[/time][set]
```

If today is Monday, January 1, 2001,

```
[scratch name=now interpolate=0]
[scratch name=now interpolate=1]
```

the first line yields

```
[time]%A, %B %d, %Y[/time]
```

while the second yields

```
Monday, January 1, 2001
```

1.1.4.2. reparse

If true ("reparse=1"), the server will process any tags in the text output by the reparsed tag.

Reparse applies only to container tags (those with an end tag). The `interpolate` attribute controls reparsing of the output of standalone tags (see above).

Most container tags will have their output re-parsed for more Interchange tags by default. If you wish to inhibit this behavior, you must explicitly set the attribute **reparse** to 0. Note that you will almost always want the default action. The only container ITL tag that doesn't have reparse set by default is [[mvasp](#)].

Example:

Assuming that name is 'Kilroy',

```
1.  [perl reparse=0]
    my $tagname = 'value';
    return "\[$tagname name\] was here\n"
[/perl]

2.  [perl reparse=1]
    my $tagname = 'value';
    return "\[$tagname name\] was here\n"
[/perl]
```

expands to

```
1.  [value name] was here

2.  Kilroy was here
```

1.1.4.3. send

Deprecated

1.1.5. Tag-specific Attributes

Each tag may accept additional named attributes which vary from tag to tag. Please see the entry for the tag in question for details about tag-specific attributes.

1.2. Looping tags and Sub-tags

Certain tags are not standalone; these are the ones that are interpreted as part of a surrounding looping tag like [[loop](#)], [[item-list](#)], [[query](#)], or [[region](#)].

[[PREFIX-accessories](#)]

[[PREFIX-alternate](#)]

1.1. Tag Syntax

Interchange Tags Reference

[\[PREFIX-calc\]](#)
[\[PREFIX-change\]](#)
[\[PREFIX-change\]](#)
[\[PREFIX-code\]](#)
[\[PREFIX-data\]](#)
[\[PREFIX-description\]](#)
[\[PREFIX-discount\]](#)
[\[PREFIX-field\]](#)
[\[PREFIX-increment\]](#)
[\[PREFIX-last\]](#)
[\[PREFIX-match\]](#)
[\[PREFIX-modifier\]](#)
[\[PREFIX-next\]](#)
[\[PREFIX-param\]](#)
[\[PREFIX-price\]](#)
[\[PREFIX-quantity\]](#)
[\[PREFIX-subtotal\]](#)
[\[if-PREFIX-data\]](#)
[\[if-PREFIX-field\]](#)
[\[modifier-name\]](#)
[\[quantity-name\]](#)

PREFIX represents the prefix that is used in that looping tag. They are only interpreted within their container and only accept positional parameters. The default prefixes:

<i>Tag</i>	<i>Prefix</i>	<i>Examples</i>
[loop]	loop	[loop-code], [loop-field price], [loop-increment]
[item-list]	item	[item-code], [item-field price], [item-increment]
[search-list]	item	[item-code], [item-field price], [item-increment]
[query]	sql	[sql-code], [sql-field price], [sql-increment]

Sub-tag behavior is consistent among the looping tags.

There are two types of looping lists; ARRAY and HASH.

An array list is the normal output of a [\[query\]](#), a search, or a [\[loop\]](#) tag. It returns from 1 to N return fields, defined in the mv_return_fields or rf variable or implicitly by means of a SQL field list. The two queries below are essentially identical:

```
[query sql="select foo, bar from products"]
[/query]

[loop search="
    ra=yes
    fi=products
    rf=foo,bar
"]
```

Both will return an array of arrays consisting of the foo column and the bar column. The Perl data structure would look like:

```
[
  ['foo0', 'bar0'],
  ['foo1', 'bar1'],
  ['foo2', 'bar2'],
  ['fooN', 'barN'],
]
```

A hash list is the normal output of the [item-list] tag. It returns the value of all return fields in an array of hashes. A normal [item-list] return might look like:

```
[
  {
    code      => '99-102',
    quantity => 1,
    size      => 'XL',
    color     => 'blue',
    mv_ib     => 'products',
  },
  {
    code      => '00-341',
    quantity => 2,
    size      => undef,
    color     => undef,
    mv_ib     => 'products',
  },
]
```

You can also return hash lists in queries:

```
[query sql="select foo, bar from products" type=hashref]
[/query]
```

Now the data structure will look like:

```
[
  { foo => 'foo0', bar => 'bar0' },
  { foo => 'foo1', bar => 'bar1' },
  { foo => 'foo2', bar => 'bar2' },
  { foo => 'fooN', bar => 'barN' },
]
```

1.2.1. [PREFIX-accessories arglist]

The same as the [accessories ...] tag except always supplied the current item code. If the list is a hash list, i.e. an [item-list], then the value of the current item hash is passed so that a value default can be established.

1.2.2. [PREFIX-alternate N] DIVISIBLE [else] NOT DIVISIBLE [/else][PREFIX-alternate]

Set up an alternation sequence. If the item-increment is divisible by `N`, the text will be displayed. If an `[else]NOT DIVISIBLE TEXT[/else]` is present, then the NOT DIVISIBLE TEXT will be displayed.

For example:

```
[item-alternate 2]EVEN[else]ODD[/else][item-alternate]
```

```
[item-alternate 3]BY 3[else]NOT by 3[/else][[/item-alternate]
```

1.2.3. [PREFIX-calc] 2 + [item-field price] [/PREFIX-calc]

Calls perl via the equivalent of the [calc] [/calc] tag pair. Much faster to execute.

1.2.4. [PREFIX-change][conditoon] ... [/condition] TEXT [/PREFIX-change]

Sets up a breaking sequence that occurs when the contents of [condition] [/condition] change. The most common one is a category break to nest or place headers.

The region is only output when a field or other repeating value between [condition] and [/condition] changes its value. This allows indented lists similar to database reports to be easily formatted. The repeating value must be a tag interpolated in the search process, such as [PREFIX-field field] or [PREFIX-data database field]. If you need access to ITL tags, you can use [PREFIX-calc] with a \$Tag->foo() call.

Of course, this will only work as you expect when the search results are properly sorted.

The value to be tested is contained within a `[condition]value[/condition]` tag pair. The `[PREFIX-change]` tag also processes an `[else] [/else]` pair for output when the value does not change.

Here is a simple example for a search list that has a field `category` and `subcategory` associated with each item:

```
<TABLE>
<TR><TH>Category</TH><TH>Subcategory</TH><TH>Product</TH></TR>
[search-list]
<TR>
  <TD>
    [item-change cat]

    [condition][item-field category][condition]

    [item-field category]
  [else]
    &nbsp;
  [/else]
  [/item-change cat]
</TD>
  <TD>
    [item-change sub]

    [condition][item-field subcategory][condition]

    [item-field subcategory]
  [else]
    &nbsp;
  [/else]
  [/item-change sub]
</TD>
  <TD> [item-field name] </TD>
[/search-list]
</TABLE>
```

The above should put out a table that only shows the category and subcategory once, while showing the name for every product. (The ` ` will prevent blanked table cells if you use a border.)

1.2.5. [PREFIX-code]

The key or code of the current loop. In an `[item-list]` this is always the product code; in a loop list it is the value of the current argument; in a search it is whatever you have defined as the first `mv_return_field` (rf).

1.2.6. [PREFIX-data table field]

Calls the column `field` in database table `table` for the current `[PREFIX-code]`. This may or may not be equivalent to `[PREFIX-field field]` depending on whether your search table is defined as one of the `ProductFiles`.

1.2.7. [PREFIX-description]

The description of the current item, as defined in the `catalog.cfg` directive `DescriptionField`. In the demo, it would be the value of the field `description` in the table `products`.

If the list is a hash list, and the lookup of `DescriptionField` fails, then the attribute `description` will be substituted. This is useful to supply shopping cart descriptions for on-the-fly items.

1.2.8. [PREFIX-discount]

The price of the current item is calculated, and the difference between that price and the list price (quantity one) price is output. This may have different behavior than you expect if you set the `[discount]` `[/discount]` tag along with quantity pricing.

1.2.9. [PREFIX-field]

Looks up a field value for the current item in one of several places, in this order:

1. The first `ProductFiles` entry.
2. Additional `ProductFiles` in the order they occur.
3. The attribute value for the item in a hash list.
4. Blank

A common user error is to do this:

```
[loop search="
    fi=foo
    se=bar
"]

[loop-field foo_field]
[/loop]
```

In this case, you are searching the table `foo` for a string of `bar`. When it is found, you wish to display the value of `foo_field`. Unless `foo` is in `ProductFiles` and the code is not present in a previous product file, you will get a blank or some value you don't want. What you really want is `[loop-data foo foo_field]`, which specifically addresses the table `foo`.

1.2.10. [PREFIX-increment]

The current count on the list, starting from either 1 in a zero-anchored list like `[loop]` or `[item-list]`, or from the match count in a search list.

If you skip items with `[PREFIX-last]` or `[PREFIX-next]`, the count is NOT adjusted.

1.2.11. [PREFIX-last] CONDITION [/PREFIX-last]

If CONDITION evaluates true (a non-whitespace value that is not specifically zero) then this will be the last item displayed.

1.2.12. [PREFIX-modifier attribute]

If the item is a hash list (i.e. `[item-list]`), this will return the value of the `attribute`.

1.2.13. [PREFIX-next] CONDITION [/PREFIX-next]

If CONDITION evaluates true (a non-whitespace value that is not specifically zero) then this item is skipped.

1.2.14. [PREFIX-param name]

1.2.15. [PREFIX-param N]

Returns the array parameter associated with the looping tag row. Each looping list returns an array of `return fields`, set in searches with `mv_return_field` or `rf`. The default is only to return the code of the search result, but by setting those parameters you can return more than one item.

In a `[query ...]` ITL tag you can select multiple return fields with something like:

```
[query prefix=prefix sql="select foo, bar from baz where foo=buz"]
  [prefix-code] [prefix-param foo] [prefix-param bar]
[/query]
```

In this case, `[prefix-code]` and `[prefix-param foo]` are synonymns, for `foo` is the first returned parameter and becomes the code for this row. Another synonym is `[prefix-param 0]`; and `[prefix-param 1]` is the same as `[prefix-param bar]`.

1.2.16. [PREFIX-price]

The price of the current code, formatted for currency. If Interchange's pricing routines cannot determine the price (i.e. it is not a valid product or on-the-fly item) then zero is returned. If the list is a hash list, the price will be modified by its `quantity` or other applicable attributes (like `size` in the demo).

1.2.17. [PREFIX-quantity]

The value of the `quantity` attribute in a hash list. Most commonly used to display the quantity of an item in a shopping cart `[item-list]`.

1.2.18. [PREFIX-subtotal]

The [PREFIX-quantity] times the [PREFIX-price]. This does take discounts into effect.

1.2.19. [if-PREFIX-data table field] IF text [else] ELSE text [/else] [/if-PREFIX-data]

Examines the data field, i.e. [PREFIX-data table field], and if it is non-blank and non-zero then the IF text will be returned. If it is false, i.e. blank or zero, the ELSE text will be returned to the page.

This is much more efficient than the otherwise equivalent `[if type=data term=table::field:: [PREFIX-code]]`.

You cannot place a condition; i.e. [if-PREFIX-data table field eq 'something']. Use `[if type=data . . .]` for that.

Careful, a space is not a false value!

1.2.20. [if-PREFIX-field field] IF text [else] ELSE text [/else] [/if-PREFIX-field]

Same as [if-PREFIX-data ...] except uses the same data rules as [PREFIX-field].

1.2.21. [modifier-name attribute]

Outputs a variable name which will set an appropriate variable name for setting the attribute in a form (usually a shopping cart). Outputs for successive items in the list:

```
1. attribute0
2. attribute1
3. attribute2
```

etc.

1.2.22. [quantity-name]

Outputs for successive items in the list:

```
1. quantity0
2. quantity1
3. quantity2
```

etc. [modifier-name quantity] would be the same as [quantity-name].

2. Tags

Each ITL tag is show below. Calling information is defined for the main tag, sub-tags are described in Sub-tags.

2.1. accessories

Allows access to Interchange's product option attributes (e.g., to choose or access product options such as a shirt's size or color).

Can build selection objects (radio, check, select boxes, etc) or forms, or can simply return a value.

Or more— see also [Item Attributes](#) and [Looping tags and Sub-tags](#).

2.1.1. Summary

```
[accessories code arg other named attributes]
[accessories code=os28044 arg=color other named attributes]
```

<i>Parameters</i>	<i>Description</i>	<i>Default</i>
code	Value of the master key in the product (or specified other) table	<i>none</i>
arg	Positionally interpreted comma-delimited list of values for the following attributes "attribute, type, column, table, name, outboard, passed" <ul style="list-style-type: none">• These may also be set individually as separate named attributes• See Item Attributes for detail	<i>none</i>

<i>Attributes</i>	<i>Default</i>
attribute	<i>none</i>
type	select
column	<i>none</i>
table	products
name	<i>none</i>
outboard	<i>none</i>
passed	<i>none</i>
key (alias for code)	<i>see code parameter above</i>
row (alias for code)	<i>see code parameter above</i>
base (alias for table)	<i>see table</i>
database (alias for table)	<i>see table</i>
db (alias for table)	<i>see table</i>
col (alias for column)	<i>see column</i>
field (alias for column}}	<i>see column</i>

Interchange Tags Reference

prepend	<i>none</i>
append	<i>none</i>
delimiter	<i>none</i>
item	<i>none</i>
extra	<i>none</i>
js	<i>none</i>
rows	<i>none</i>
cols	<i>none</i>
width	<i>none</i>
default	<i>none</i>
price	<i>none</i>
contains (type = radio or checkbox)	<i>none</i>
joiner (type = hyperlink)	<i>none</i>
href (type = hyperlink)	<i>none</i>
template (type = hyperlink)	<i>none</i>
selected begin (type = hyperlink)	<i>none</i>
form (type = hyperlink)	<i>none</i>
empty (type = hyperlink)	<i>none</i>
secure (type = hyperlink)	<i>none</i>
new (type = combo or reverse_combo)	<i>none</i>
interpolate (reparse)	<i>No</i>

<i>Other_Charactreristics</i>	
Invalidates cache	<i>No</i>
Has Subtags	<i>No</i>
Container tag	<i>No</i>

Tag expansion example:

```
[accessories os28044 color]
```

This will search the product database for a field named "color". If an entry "beige=Almond, gold=Harvest Gold, White*, green=Avocado" is found, a select box like this will be built:

```
<SELECT NAME="mv_order_color">
<OPTION VALUE="beige">Almond
<OPTION VALUE="gold">Harvest Gold
<OPTION SELECTED>White
<OPTION VALUE="green">Avocado
</SELECT>
```

ASP-like Perl call:

```
$Tag->accessories( { code    => '[[EXAMPLE_SKU]]',  
                  arg      => 'color, radio,',  
                  table    => 'special_products', } );
```

or similarly with positional parameters,

```
$Tag->accessories($code, $arg, $attribute_hash_reference);
```

2.1.1.1. See Also

[Item Attributes](#) and [Looping tags and Sub-tags](#).

2.1.2. Description

Allows access to Interchange's product option attributes (e.g., to choose or access product options such as a shirt's size or color).

If passed any of the optional arguments, initiates special processing of item attributes based on entries in the product database.

Interchange allows item attributes to be set for each ordered item. This allows a size, color, or other modifier to be attached to a line item in the shopping cart. Previous attribute values can be resubmitted by means of a hidden field on a form.

The `catalog.cfg` file directive *UseModifier* is used to set the name of the modifier or modifiers. For example

```
UseModifier      size color
```

will attach both a size and color attribute to each item code that is ordered.

IMPORTANT NOTE: You may not use the following names for attributes:

```
item group quantity code mv_ib mv_mi mv_si
```

You can also set it in scratch with the `mv_UseModifier` scratch variable — `[set mv_UseModifier]size color[/set]` has the same effect as above. This allows multiple options to be set for products. Whichever one is in effect at order time will be used. Be careful, you cannot set it more than once on the same page. Setting the `mv_separate_items` or global directive *SeparateItems* places each ordered item on a separate line, simplifying attribute handling. The scratch setting for `mv_separate_items` has the same effect.

The modifier value is accessed in the [\[item-list\]](#) loop with the `[item-param attribute]` tag, and form input fields are placed with the `[modifier-name attribute]` tag. This is similar to the way that quantity is handled.

NOTE: You must be sure that no fields in your forms have digits appended to their names if the variable is the same name as the attribute name you select, as the `[modifier-name size]` variables will be placed in the user session as the form variables `size0`, `size1`, `size2`, etc.

Interchange Tags Reference

Interchange will automatically generate the select boxes when the `[accessories <code size>]` or `[item-accessories size]` tags are called. They have the syntax:

```
[item_accessories attribute, type*, column*, table*, name*, outboard*]

[accessories code=sku
  attribute=modifier
  type="select|radio|display|show|checkbox|text|textarea"*
  column=column_name*
  table=db_table*
  name=varname
  outboard=key
  passed="value=label, value2, value3=label 3" ]
[accessories
  js=| onChange="set_description(simple_options, variant);" |
  type=select
  name="[item-param o_group]"
  passed="---choose---,[item-param o_value]"
]
```

2.1.2.1. code

This is the master key (usually SKU) of the specified table, or the products table by default. The code argument is implicit if looping on `[item_accessories]`.

2.1.2.2. arg

The optional `arg` argument

2.1.2.3. attribute

2.1.2.4. type

2.1.2.5. column

2.1.2.6. table

2.1.2.7. name

2.1.2.8. outboard

2.1.2.9. passed

2.1.2.10. prepend

2.1.2.11. append

2.1.2.12. delimiter

2.1.2.13. item

2.1.2.14. extra

2.1.2.15. js

2.1.2.16. rows

2.1.2.17. cols

2.1.2.18. width

2.1.2.19. default

2.1.2.20. price

2.1.2.21. contains (type = radio or checkbox)

2.1.2.22. joiner (type = hyperlink)

2.1.2.23. href (type = hyperlink)

2.1.2.24. template (type = hyperlink)

2.1.2.25. selected_begin (type = hyperlink)

2.1.2.26. form (type = hyperlink)

2.1.2.27. empty (type = hyperlink)

2.1.2.28. secure (type = hyperlink)

2.1.2.29. new (type = combo or reverse_combo)

2.1.2.30. Examples

Tag expansion example:

```
[accessories code=os28044 arg=color]
```

```
TAGRESULT
```

Positional call example:

```
[accessories code arg]
```

Named call example:

```
[accessories code=[[EXAMPLE_SKU]] arg=color]
```

2.1.2.31. code

Not needed for item–accessories, this is the product code of the item to reference.

2.1.2.32. attribute

The item attribute as specified in the UseModifier configuration directive. Typical are `size` or `color`.

2.1.2.33. type

The action to be taken. One of:

<i>Action</i>	<i>Description</i>
select	Builds a dropdown <SELECT> menu for the attribute. Note: This is the default.
multiple	Builds a multiple dropdown <SELECT> menu for the attribute. The size is equal to the number of option choices.
display	Shows the label text for *only the selected option*.
show	Shows the option choices (no labels) for the option.
radio	Builds a radio box group for the item, with spaces separating the elements.
radio nbsp	Builds a radio box group for the item, with separating the elements.
radio left n	Builds a radio box group for the item, inside a table, with the checkbox on the left side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns.
radio right n	Builds a radio box group for the item, inside a table, with the checkbox on the right side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns.
check	Builds a checkbox group for the item, with spaces separating the elements.
check nbsp	Builds a checkbox group for the item, with separating the elements.
check left n	Builds a checkbox group for the item, inside a table, with the checkbox on the left side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns.
check right n	Builds a checkbox group for the item, inside a table, with the checkbox on the right side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns.
textarea_XX_YY	A textarea with XX columns and YY rows
text_XX	A text box with XX size in characters
combo	Special type, used with nullselect filter, for selecting from a list or inputting a new value
reverse_combo	Special type, used with last_non_null filter, for selecting from a list or inputting a new value — differs from combo in order of presentation
move_combo	Special type, used with null_to_space or null_to_comma filter, for selecting multiple non-ordered values from a list or inputting into a textarea
links	Produces a series of links based on the option values. The base form value is passed via the form parameter, just like in an [area ...] or [page ...] tag, and the value is named with the passed NAME attribute.

The default is 'select', which builds an HTML select form entry for the attribute. Also recognized is 'multiple', which generates a multiple-selection drop down list, 'show', which shows the list of possible attributes, and 'display', which shows the label text for the selected option only.

2.1.2.34. column

The database column name to be used to build the entry (usually a field in the products database). Defaults to a field named the same as the attribute.

2.1.2.35. table

The database table to find **column** in, defaults to the first products file where the item code is found.

2.1.2.36. name

Name of the form variable to use if a form is being built. Defaults to **mv_order_attribute** — i.e. if the attribute is **size**, the form variable will be named **mv_order_size**. If the variable is set in the user session, the widget will "remember" its previous setting.

2.1.2.37. outboard

If calling the item-accessories tag, and you wish to select from an outboard database table with a different key from the item code, you can pass the key to use to find the accessory data.

2.1.2.38. passed

Options for the widget if set explicitly, i.e. overrides a database fetch of the attribute input string.

For example, to generate a select box with a blank option (perhaps forcing a select), the value of blue with a label of **Blue**, and the value of green with a label of **Sea Green**, do:

```
[accessories
  type=select
  name=color
  passed="---select--, blue=Blue, green=Sea Green"
]
```

This will generate:

```
<SELECT NAME="color">
<OPTION VALUE=" ">>---select--
<OPTION VALUE="blue">Blue
<OPTION VALUE="green">Sea Green
</SELECT>
```

2.1.2.39. href

Base HREF for the link in a **links** type. Default is the current page.

2.1.2.40. form

Base value for the form in a **links** type. Default is **mv_action=return**, which will simply set the variable value in the link.

For example, to generate a series of links that set the variable "color" to the same as the options described in the **passed** attribute (blank, blue, or green), do

Interchange Tags Reference

```
[accessories
  type=links
  name=color
  passed="--select--, blue=Blue, green=Sea Green"
]
```

This will generate:

```
<A HREF="VENDURL/MV_PAGE?mv_action=return&color=blue">Blue</A><BR>
<A HREF="VENDURL/MV_PAGE?mv_action=return&color=green">Sea Green</A>
```

where VENDURL is your Interchange URL for the catalog
MV_PAGE is the current page

If you want the empty "--select—" option to show up, pass an empty=1 parameter.

When called with an attribute, the database is consulted and looks for a comma-separated list of attribute options. They take the form:

```
name=Label Text, name=Label Text*
```

The label text is optional -- if none is given, the **name** will be used.

If an asterisk is the last character of the label text, the item is the default selection. If no default is specified, the first will be the default. An example:

```
[item_accessories color]
```

This will search the product database for a field named "color". If an entry "beige=Almond, gold=Harvest Gold, White*, green=Avocado" is found, a select box like this will be built:

```
<SELECT NAME="mv_order_color">
<OPTION VALUE="beige">Almond
<OPTION VALUE="gold">Harvest Gold
<OPTION SELECTED>White
<OPTION VALUE="green">Avocado
</SELECT>
```

In combination with the mv_order_item and mv_order_quantity variables this can be used to allow entry of an attribute at time of order.

If used in an item list, and the user has changed the value, the generated select box will automatically retain the current value the user has selected.

The value can then be displayed with [item-modifier size] on the order report, order receipt, or any other page containing an [\[item-list\]](#).

When called with an attribute, the database is consulted and looks for a comma-separated list of attribute options. They take the form:

```
name=Label Text, name=Label Text*
```

The label text is optional -- if none is given, the **name** will be used.

Interchange Tags Reference

If an asterisk is the last character of the label text, the item is the default selection. If no default is specified, the first will be the default. An example:

```
[accessories TK112 color]
```

This will search the product database for a field named "color". If an entry "beige=Almond, gold=Harvest Gold, White*, green=Avocado" is found, a select box like this will be built:

```
<SELECT NAME="mv_order_color">
<OPTION VALUE="beige">Almond
<OPTION VALUE="gold">Harvest Gold
<OPTION SELECTED>White
<OPTION VALUE="green">Avocado
</SELECT>
```

In combination with the *mv_order_item* and *mv_order_quantity* variables this can be used to allow entry of an attribute at time of order.

2.1.2.41. Emulating with a loop

Below is a fragment from a shopping basket display form which shows a selectable size with "sticky" setting and a price that changes based upon the modifier setting. Note that this would normally be contained within the `[item_list]` `[/item-list]` pair.

```
<SELECT NAME="[modifier-name size]">
[loop option="[modifier-name size]" list="S, M, L, XL"]
<OPTION> [loop-code] -- [price code="[item-code]" size="[loop-code]"]
[/loop]
</SELECT>
```

The above is essentially the same as would be output with the `[item-accessories size]` tag if the product database field `size` contained the value `S, M, L, XL`, but contains the adjusted price.

2.2. and

2.2.1. Summary

Parameters: **type term op compare**

THIS TAG HAS SPECIAL POSITIONAL PARAMETER HANDLING.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

Called Routine for positional:

ASP/perl tag calls:

```
$Tag->and(
{
  type => VALUE,
  term => VALUE,
  op => VALUE,
  compare => VALUE,
}
)
```

OR

```
$Tag->and($type, $term, $op, $compare);
```

Attribute aliases

```
base ==> type
comp ==> compare
operator ==> op
```

2.2.2. Description

The [and ...] tag is only used in conjunction with [if ...]. Example:

```
[if value fname]
[and value lname]
Both first and last name are present.
[else]
Missing one of "fname" and "lname" from $Values.
[/else]
[/if]
```

See [if ...].

2.3. area

Alias: **href**

Expands to the URL for an Interchange page or action, including the Interchange session ID and supplied arguments. This is very similar to the [page](#) tag -- these are equivalent:

```
[page href=dir/page arg=mv_arg]TargetName[/page]
<A HREF="[area href=dir/page arg=mv_arg]">TargetName</A>
```

2.3.1. Summary

```
[area href arg other named attributes]
[area href=dir/page arg=page_arguments other named attributes]
```

<i>Parameters</i>	<i>Description</i>	<i>Default</i>
href	Path to Interchange page or action <i>Special arguments</i> <ul style="list-style-type: none"> ◆ 'scan' links to a search (using search arguments in arg) ◆ 'http://...' external link (requires 	process

Interchange Tags Reference

	form attribute)	
arg	Interchange arguments to page or action	<i>none</i>

<i>Attributes</i>	<i>Default</i>
form	<i>none</i>
search	<i>No</i>
secure	<i>No</i>
interpolate (reparse)	<i>No</i>

<i>Other_Characteristics</i>	
Invalidates cache	<i>No</i>
Macro	<i>No</i>
Has end tag	<i>No</i>

Tag expansion example:

```
[area href=dir/page.html arg="arg1=AA/arg2=BB"]  
  
www.here.com/cgi-bin/mycatalog/page.html?mv_session_id=6CZ2whqo&\  
mv_pc=1&mv_arg=arg1%3dAA/arg2%3dBB
```

ASP-like Perl call:

```
$Tag->area( { href => "dir/page",  
             arg  => "arguments", } );
```

or similarly with positional parameters,

```
$Tag->area($href, $arg, $attribute_hash_reference);
```

2.3.1.1. See Also

[page](#)

2.3.2. Description

The `area` tag is very similar to the [page](#) tag. It produces the URL to call an Interchange page, but it differs from `page` in that it does not supply the surrounding `<A HREF ...>` notation. This can be used to get control of your HREF items, perhaps to place an ALT string or a Javascript construct.

It was originally named `area` because it also can be used in a client-side image map.

The `area` tag has an alias of `href`. The two links below are identical in operation:

```
<A HREF="[area href=catalog]" ALT="Main catalog page">Catalog Home</A>  
<A HREF="[href href=catalog]" ALT="Main catalog page">Catalog Home</A>
```

The optional *arg* is used just as in the [page](#) tag.

2.3.2.1. form

The optional *form* argument allows you to encode a form in the link.

```
<A HREF="[area form="mv_order_item=os28044
mv_order_size=15oz
mv_order_quantity=1
mv_separate_items=1
mv_todo=refresh"]"> Order 15oz Framing Hammer</A>
```

See the description of the [page](#) tag for more detail.

2.3.2.2. search

Interchange allows you to pass a search in a URL. There are two ways to do this:

1. Place the search specification in the named *search* attribute.
 - ◆ Interchange will ignore the *href* parameter (the link will be set to 'scan').
 - ◆ If you give the *arg* parameter a value, that value will be available as [[value](#) *mv_arg*] within the search display page.
2. Set the *href* parameter to 'scan' and set *arg* to the search specification.
 - ◆ Note that you can use this form positionally — the values go into *href* and *arg*, so you do not have to name parameters.

These are identical:

```
<A HREF="[area scan
se=Impressionists
sf=category]">Impressionist Paintings</A>

<A HREF="[area href=scan
arg="se=Impressionists
sf=category]">Impressionist Paintings</A>

<A HREF="[area search="se=Impressionists
sf=category]">Impressionist Paintings</A>
```

See the description of the [page](#) tag for more detail.

2.3.2.3. Examples

Tag expansion example:

```
[area href=dir/page.html arg="arg1=AA/arg2=BB"]

www.here.com/cgi-bin/mycatalog/page.html?mv_session_id=6CZ2whqo&\
mv_pc=1&mv_arg=arg1%3dAA/arg2%3dBB
```

Positional call example:

```
<A HREF="[area ord/basket]">Check basket</A>
```

Named call example:

```
<A HREF="[area ord/basket]">Check basket</A>
```

HTML-embedded example (disabled if `no_html_parse` [Pragma](#) active):

```
<A MV="area dir/page" HREF="dir/page.html">
```

2.4. attr_list

2.4.1. Summary

Parameters: **hash**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[attr_list] FOO [/attr_list]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->attr_list(
    {
        hash => VALUE,
    },
    BODY
)
```

OR

```
$Tag->attr_list($hash, $BODY);
```

2.4.2. Description

Tags an attribute list with values from a hash. Designed for use in embedded Perl.

Example:

```
[perl tables=products]
    my %opt = (
        hashref => 1,
        sql => 'select * from products',
    );
    my $ary_of_hash = $Db{products}->query(\%opt);
    my $template = <<EOF;
{sku} - {description} - {price|Call for price}
```

Interchange Tags Reference

```
{image?}<IMG SRC="{image}">{/image?}
{image:}No image available{/image:}
EOF
        foreach my $ref (@$ary_of_hash) {
            $out .= $Tag->attr_list($template, $ref);
        }
        return $out;
[/perl]
```

Tags according to the following rules:

2.4.2.1. {key}

Inserts the value of the key for the reference. In a database query, this is the column name.

2.4.2.2. {key|fallback string}

Displays the value of {key} or if it is zero or blank, the fallback string.

2.4.2.3. {key true string}

Displays `true string` if the value of {key} is non-blank, non-zero, or displays nothing if the key is false.

2.4.2.4. {key?} true text {/key?}

Displays `true text` if the value of {key} is non-blank, non-zero, and nothing otherwise.

2.4.2.5. {key:} false text {/key:}

Displays `false text` if the value of {key} is blank or zero, and nothing otherwise.

2.5. banner

2.5.1. Summary

Parameters: **category**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here.

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->banner(
{
```



```

        category => VALUE,
    }
)

OR

$Tag->banner($category, $ATTRHASH);

```

2.5.2. Description

The [banner ...] tag is designed to implement random or rotating banner displays in your Interchange pages. See the main Interchange documentation, section *Banner/Ad rotation*.

2.6. bounce

2.6.1. Summary

Parameters: **href if**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```

$Tag->bounce(
    {
        href => VALUE,
        if => VALUE,
    }
)

OR

$Tag->bounce($href, $if);

```

2.6.2. Description

The [bounce ...] tag is designed to send an HTTP redirect (302 status code) to the browser and redirect it to another (possibly Interchange-parsed) page.

It will stop ITL code execution at that point; further tags will not be run through the parser. Bear in mind that if you are inside a looping list, that list will run to completion and the [bounce] tag will not be seen until the loop is complete.

Example of bouncing to an Interchange parsed page:

```
[if !scratch real_user]
[bounce href="[area violation]"]
[/if]
```

Note the URL is produced by the [area . . .] ITL tag.

Since the HTTP says the URL needs to be absolute, this one might cause a browser warning:

```
[if value go_home]
[bounce href="/"]
[/if]
```

But running something like one of the Interchange demos you can do:

```
[if value go_home]
[bounce href="__SERVER_NAME__/" ]
[/if]

[if value go_home]
[bounce href="/"]
[/if]
```

2.7. calc

2.7.1. Summary

No parameters.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [calc] FOO [/calc]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->calc(
{
},
BODY
)
```

OR

```
$Tag->calc($BODY);
```

2.7.2. Description

syntax: [calc] EXPRESSION [/calc]

Starts a region where the arguments are calculated according to normal arithmetic symbols. For instance:

```
[calc] 2 + 2 [/calc]
```

will display:

4

The [calc] tag is really the same as the [perl] tag, except that it doesn't accept arguments, interpolates surrounded Interchange tags by default, and is slightly more efficient to parse.

TIP: The [calc] tag will remember variable values inside one page, so you can do the equivalent of a memory store and memory recall for a loop.

ASP NOTE: There is never a reason to use this tag in a [perl] or ASP section.

2.8. cart

2.8.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->cart(  
    {  
        name => VALUE,  
    }  
)
```

OR

```
$Tag->cart($name);
```

2.8.2. Description

Sets the name of the current shopping cart for display of shipping, price, total, subtotal, shipping, and nitems tags.

2.9. catch

2.9.1. Summary

Parameters: **label**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[catch] FOO [/catch]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->catch(
    {
        label => VALUE,
    },
    BODY
)
```

OR

```
$Tag->catch($label, $ATTRHASH, $BODY);
```

2.9.2. Description

NO Description

2.10. cgi

2.10.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->cgi(
    {
        name => VALUE,
    }
)
```

OR

```
$Tag->cgi($name);
```

2.10.2. Description

Displays the value of a CGI variable **submitted to the current page**. This is similar to [value ...], except it displays the transitory values that are submitted with every request.

For instance, if you access the following URL:

```
http://VENDURL/pagename?foo=bar
```

bar will be substituted for [cgi foo].

This is the same as \$CGI->{foo} in embedded Perl.

2.11. checked

2.11.1. Summary

Parameters: **name value**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->checked(
    {
        name => VALUE,
        value => VALUE,
    }
)
```

OR

```
$Tag->checked($name, $value, $ATTRHASH);
```

2.11.2. Description

You can provide a "memory" for drop-down menus, radio buttons, and checkboxes with the [checked] and [selected] tags.

```
<INPUT TYPE=radio NAME=foo
      VALUE=on [checked name=foo value=on default=1]>
<INPUT TYPE=radio NAME=foo
      VALUE=off [checked name=foo value=off]>
```

This will output CHECKED if the variable var_name is equal to value. Not case sensitive unless the optional case=1 parameter is used.

The default parameter, if true (non-zero and non-blank), will cause the box to be checked if the variable has never been defined.

Note that CHECKBOX items will never submit their value if not checked, so the box will not be reset. You must do something like:

```
<INPUT TYPE=checkbox NAME=foo
      VALUE=1 [checked name=foo value=1 default=1]>
[value name=foo set=""]
```

By default, the Values space (i.e. [value foo]) is checked — if you want to use the volatile CGI space (i.e. [cgi foo]) use the option cgi=1.

2.12. control

2.12.1. Summary

Parameters: **name default**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here.

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->control(
{
  name => VALUE,
  default => VALUE,
}
```

```
)
```

OR

```
$Tag->control($name, $default, $ATTRHASH);
```

2.12.2. Description

NO Description

2.13. control_set

2.13.1. Summary

Parameters: **index**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [control_set] FOO [/control_set]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->control_set(
    {
        index => VALUE,
    },
    BODY
)
```

OR

```
$Tag->control_set($index, $ATTRHASH, $BODY);
```

2.13.2. Description

NO Description

2.14. counter

2.14.1. Summary

Parameters: **file**

Positional parameters in same order.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->counter(  
    {  
        file => VALUE,  
    }  
)
```

OR

```
$Tag->counter($file, $ATTRHASH);
```

Attribute aliases

```
name ==> file
```

2.14.2. Description

Manipulates a persistent counter, by default incrementing it and returning the new value.

The counter value is stored in the specified file. If the file name begins with a "/" then it is an absolute path. Otherwise, it is relative to VendRoot. The default file is `etc/counter`. If the file does not exist, it is created and initialized to the value of the `start` parameter.

The counter is implemented using Perl's `File::Counter` module, which protects the file against simultaneous access by multiple processes.

WARNING: This tag will not work under Safe, i.e. in embedded Perl.

Additional parameters:

2.14.2.1. **decrement=1**

Causes the counter to count down instead of up.

2.14.2.2. **start=50**

Causes a new counter to be created and to start from 50 (for example) if it did not exist before.

2.14.2.3. **value=1**

Shows the value of the counter without incrementing or decrementing it.

2.15. currency

2.15.1. Summary

Parameters: **convert noformat**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [currency] FOO [/currency]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->currency(
    {
        convert => VALUE,
        noformat => VALUE,
    },
    BODY
)
```

OR

```
$Tag->currency($convert, $noformat, $BODY);
```

2.15.2. Description

When passed a value of a single number, formats it according to the currency specification. For instance:

```
[currency]4[/currency]
```

will display:

```
4.00
```

or something else depending on the *Locale* and *PriceCommas* settings. It can contain a [calc] region. If the optional "convert" parameter is set, it will convert the value according to *PriceDivide*> for the current locale. If *Locale* is set to *fr_FR*, and *PriceDivide* for *fr_FR* is 0.167, the following sequence

```
[currency convert=1] [calc] 500.00 + 1000.00 [/calc] [/currency]
```

will cause the number 8.982,04 to be displayed.

2.16. data

2.16.1. Summary

Parameters: **table field key**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here.

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->data(
    {
        table => VALUE,
        field => VALUE,
        key => VALUE,
    }
)
```

OR

```
$Tag->data($table, $field, $key, $ATTRHASH);
```

Attribute aliases

```
base ==> table
code ==> key
col ==> field
column ==> field
database ==> table
name ==> field
row ==> key
```

2.16.2. Description

Syntax: `[data table=db_table column=column_name key=key filter="uc|lc|name|namecase|no_white|etc."* append=1* value="value to set to"* increment=1*]`

Returns the value of the field in a database table, or (DEPRECATED) from the `session` namespace. If the optional **value** is supplied, the entry will be changed to that value. If the option `increment*` is present, the field will be atomically incremented with the value in **value**. Use negative numbers in `value` to decrement. The `append` attribute causes the value to be appended; and finally, the `filter` attribute is a set of Interchange filters that are applied to the data 1) after it is read; or 2) before it is placed in the table.

If a DBM-based database is to be modified, it must be flagged writable on the page calling the write tag. Use

Interchange Tags Reference

[tag flag write]products[/tag] to mark the products database writable, for example. **This must be done before ANY access to that table.**

DEPRECATED BEHAVIOR: (replace with session tag). In addition, the [data ...] tag can access a number of elements in the Interchange session database:

accesses	Accesses within the last 30 seconds
arg	The argument passed in a [page ...] or [area ...] tag
browser	The user browser string
cybercash_error	Error from last CyberCash operation
cybercash_result	Hash of results from CyberCash (access with usertag)
host	Interchange's idea of the host (modified by DomainTail)
last_error	The last error from the error logging
last_url	The current Interchange path_info
logged_in	Whether the user is logged in (add-on UserDB feature)
pageCount	Number of unique URLs generated
prev_url	The previous path_info
referer	HTTP_REFERER string
ship_message	The last error messages from shipping
source	Source of original entry to Interchange
time	Time (seconds since Jan 1, 1970) of last access
user	The REMOTE_USER string
username	User name logged in as (UserDB feature)

NOTE: Databases will hide session values, so don't name a database "session". or you won't be able to use the [data ...] tag to read them. Case is sensitive, so in a pinch you could call the database "Session", but it would be better not to use that name at all.

2.17. default

2.17.1. Summary

Parameters: **name default**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->default(  
  {  
    name => VALUE,  
    default => VALUE,  
  }  
)
```

OR

```
$Tag->default($name, $default, $ATTRHASH);
```

2.17.2. Description

Returns the value of the user form variable `variable` if it is non-empty. Otherwise returns `default`, which is the string "default" if there is no default supplied. Got that? This tag is DEPRECATED anyway.

2.18. description

2.18.1. Summary

Parameters: **code base**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->description(
    {
        code => VALUE,
        base => VALUE,
    }
)
```

OR

```
$Tag->description($code, $base);
```

2.18.2. Description

Expands into the description of the product identified by `code` as found in the products database. This is the value of the database field that corresponds to the `catalog.cfg` directive `DescriptionField`. If there is more than one products file defined, they will be searched in order unless constrained by the optional argument **base**.

This tag is especially useful for multi-language catalogs. The `DescriptionField` directive can be set for each locale and point to a different database field; for example `desc_en` for English, `desc_fr` for French, etc.

2.19. discount

2.19.1. Summary

Parameters: **code**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [discount] FOO [/discount]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->discount(
    {
        code => VALUE,
    },
    BODY
)
```

OR

```
$Tag->discount($code, $BODY);
```

2.19.2. Description

Product discounts can be set upon display of any page. The discounts apply only to the customer receiving them, and are of one of three types:

1. A discount for one particular item code (code/key is the item-code)
2. A discount applying to all item codes (code/key is ALL_ITEMS)
3. A discount applied after all items are totaled (code/key is ENTIRE_ORDER)

The discounts are specified via a formula. The formula is scanned for the variables \$q and \$s, which are substituted for with the item *quantity* and *subtotal* respectively. In the case of the item and all items discount, the formula must evaluate to a new subtotal for all items *of that code* that are ordered. The discount for the entire order is applied to the entire order, and would normally be a monetary amount to subtract or a flat percentage discount.

Discounts are applied to the effective price of the product, including any quantity discounts.

To apply a straight 20% discount to all items:

```
[discount ALL_ITEMS] $s * .8 [/discount]
```

or with named attributes:

```
[discount code=ALL_ITEMS] $s * .8 [/discount]
```

To take 25% off of only item 00–342:

```
[discount 00-342] $s * .75 [/discount]
```

To subtract \$5.00 from the customer's order:

```
[discount ENTIRE_ORDER] $s - 5 [/discount]
```

To reset a discount, set it to the empty string:

```
[discount ALL_ITEMS][discount]
```

Perl code can be used to apply the discounts. Here is an example of a discount for item code 00–343 which prices the *second* one ordered at 1 cent:

```
[discount 00-343]
return $s if $q == 1;
my $p = $s/$q;
my $t = ($q - 1) * $p;
$t .= 0.01;
return $t;
[/discount]
```

If you want to display the discount amount, use the [item–discount] tag.

```
[item-list]
Discount for [item-code]: [item-discount]
[/item-list]
```

Finally, if you want to display the discounted subtotal in a way that doesn't correspond to a standard Interchange tag, you can use the [calc] tag:

```
[item-list]
Discounted subtotal for [item-code]: [currency][calc]
                                     [item-price noformat] * [item-quantity]
                                     [/calc][currency]
[/item-list]
```

%% dump %% Prints a dump of the current user session as expanded by Data::Dumper. Includes any CGI environment passed from the server.

For an example, create a page containing <XMP>[dump]</XMP> and see what it displays.

%% either %% The [either](#)[this[or](#)that[/either] implements a check for the first non–zero, non–blank value. It splits on [or], and then parses each piece in turn. If a value returns true (in the Perl sense — non–zero, non–blank) then subsequent pieces will be discarded without interpolation.

Example:

```
[either][value must_be_here][or][bounce href="[area incomplete]"[/either]
```

Always strips leading and trailing whitespace.

2.20. discount_subtotal

2.20.1. Description

Inserts the discounted subtotal of the ordered items.

2.21. dump

2.21.1. Summary

No parameters.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->dump (
    {
    }
)
```

OR

```
$Tag->dump ( $ ) ;
```

2.21.2. Description

NO Description

2.22. ecml

2.22.1. Summary

Parameters: **name function**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->ecml(  
    {  
        name => VALUE,  
        function => VALUE,  
    }  
)
```

OR

```
$Tag->ecml($name, $function, $ATTRHASH);
```

2.22.2. Description

NO Description

2.23. either

2.23.1. Summary

No parameters.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [either] FOO [/either]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->either(  
    {  
    },  
    BODY  
)
```

OR

```
$Tag->either($BODY);
```

2.23.2. Description

NO Description

2.24. error

2.24.1. Summary

Parameters: **name**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here.

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->error(
    {
        name => VALUE,
    }
)
```

OR

```
$Tag->error($name, $ATTRHASH);
```

2.24.2. Description

```
[error var options]
    var is the error name, e.g. "session"
```

The `[error ...]` tag is designed to manage form variable checking for the Interchange submit form processing action. It works in conjunction with the definition set in `mv_order_profile`, and can generate error messages in any format you desire.

If the variable in question passes order profile checking, it will output a label, by default **bold** text if the item is required, or normal text if not (controlled by the `<require>` parameter). If the variable fails one or more order checks, the error message will be substituted into a template and the error cleared from the user's session.

(Below is as of 4.03, the equivalent in 4.02 is `[if type=explicit compare="[error all=1 keep=1]" ... [/if].`)

To check errors without clearing them, you can use the idiom:

```
[if errors]
<FONT SIZE="+1" COLOR=RED>
    There were errors in your form submission.
</FONT>
<BLOCKQUOTE>
    [error all=1 show_error=1 joiner="<BR>"]
```

```
</BLOCKQUOTE>
[ /if ]
```

The options are:

2.24.2.1. **all=1**

Display all error messages, not just the one referred to by <var>. The default is only display the error message assigned to <var>.

text=<optional string to embed the error message(s) in>

place a "%s" somewhere in 'text' to mark where you want the error message placed, otherwise it's appended on the end. This option also implies show_error.

2.24.2.2. **joiner=<char>**

Character used to join multiple error messages. Default is '\n', a newline.

2.24.2.3. **keep=1**

keep=1 means don't delete the error messages after copy; anything else deletes them.

2.24.2.4. **show_error=1**

show_error=1 means return the error message text; otherwise just the number of errors found is returned.

2.24.2.5. **show_label=1**

show_label=1 causes the field label set by a previous [error] tag's std_label attribute (see below) to be included as part of the error message, like this:

```
First Name: blank
```

If no std_label was set, the variable name will be used instead. This can also be used in combination with show_var to show both the label and the variable name.

show_label was added in 4.7.0.

2.24.2.6. **show_var=1**

show_var=1 includes the name of the variable the error was found in as part of the error message, like this:

```
email: 'bob#nothing,net' not a valid email address
```

2.24.2.7. **std_label**

std_label=<label string for error message>

used with 'required' to display a standardized error format. The HTML formatting can be set via the global variable MV_ERROR_STD_LABEL with the default being:

```
<FONT COLOR=RED>label_str<SMALL><I>(%s)</I></SMALL></FONT>
```

where <label_str> is what you set std_label to and %s is substituted with the error message. This option can not be used with the text= option.

2.24.2.8. required=1

Specifies that this is a required field for formatting purposes. In the std_label format, it means the field will be bolded. If you specify your own label string, it will insert HTML anywhere you have {REQUIRED: HTML}, but only when the field is required.

2.25. export

2.25.1. Summary

Parameters: **table**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->export(  
    {  
        table => VALUE,  
    }  
)
```

OR

```
$Tag->export($table, $ATTRHASH);
```

Attribute aliases

```
base ==> table  
database ==> table
```

2.25.2. Description

NO Description

2.26. field

2.26.1. Summary

Parameters: **name code**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->field(
    {
        name => VALUE,
        code => VALUE,
    }
)
```

OR

```
$Tag->field($name, $code);
```

Attribute aliases

```
col ==> name
column ==> name
field ==> name
key ==> code
row ==> code
```

2.26.2. Description

HTML example: `<PARAM MV=field MV.COL=column MV.ROW=key>`

Expands into the value of the field *name* for the product identified by *code* as found by searching the products database. It will return the first entry found in the series of *Product Files*. the products database. If you want to constrain it to a particular database, use the `[data base name code]` tag.

Note that if you only have one ProductFile products, which is the default, `[field column key]` is the same as `[data products column key]`.

2.27. file

2.27.1. Summary

Parameters: **name type**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->file(
    {
        name => VALUE,
        type => VALUE,
    }
)
```

OR

```
$Tag->file($name, $type);
```

2.27.2. Description

Inserts the contents of the named file. The file should normally be relative to the catalog directory -- file names beginning with `/` or `..` are not allowed if the Interchange server administrator has set *NoAbsolute* to *Yes*.

The optional `type` parameter will do an appropriate ASCII translation on the file before it is sent.

2.28. filter

2.28.1. Summary

Parameters: **op**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[filter] FOO [/filter]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->filter(  
    {  
        op => VALUE,  
    },  
    BODY  
)
```

OR

```
$Tag->filter($op, $BODY);
```

2.28.2. Description

Applies any of Interchange's standard filters to an arbitrary value, or you may define your own. The filters are also available as parameters to the `cgi`, `data`, and `value` tags.

Filters can be applied in sequence and as many as needed can be applied.

Here is an example. If you store your author or artist names in the database "LAST, First" so that they sort properly, you still might want to display them normally as "First Last". This call

```
[filter op="name namecase"]WOOD, Grant[/filter]
```

will display as

```
Grant Wood
```

Another way to do this would be:

```
[data table=products column=artist key=99-102 filter="name namecase"]
```

Filters available include:

2.28.2.1. `cgi`

Returns the value of the CGI variable. Useful for starting a filter sequence with a seed value.

```
'cgi' => sub {  
    return $CGI::values(shift);  
},
```

2.28.2.2. `digits`

Returns only digits.

```
'digits' => sub {  
    my $val = shift;  
    $val =~ s/\D+//g;  
    return $val;  
},
```

2.28.2.3. digits_dot

Returns only digits and periods, i.e. [.0–9]. Useful for decommifying numbers.

```
'digits_dot' => sub {
    my $val = shift;
    $val =~ s/[^\.d.]+//g;
    return $val;
},
```

2.28.2.4. dos

Turns linefeeds into carriage–return / linefeed pairs.

```
'dos' => sub {
    my $val = shift;
    $val =~ s/\r?\n/\r\n/g;
    return $val;
},
```

2.28.2.5. entities

Changes < to <; " to "; etc.

```
'entities' => sub {
    return HTML::Entities::encode(shift);
},
```

2.28.2.6. gate

Performs a security screening by testing to make sure a corresponding scratch variable has been set.

```
'gate' => sub {
    my ($val, $var) = @_;
    return '' unless $::Scratch->{$var};
    return $val;
},
```

2.28.2.7. lc

Lowercases the text.

```
'lc' => sub {
    return lc(shift);
},
```

2.28.2.8. lookup

Looks up an item in a database based on the passed table and column. Call would be:

```
[filter op="uc lookup.country.name"]us[/filter]
```

This would be the equivalent of [data table=country column=name key=US].

```
'lookup' => sub {
```

Interchange Tags Reference

```
my ($val, $tag, $table, $column) = @_;  
return tag_data($table, $column, $val) || $val;  
},
```

2.28.2.9. mac

Changes newlines to carriage returns.

```
'mac' => sub {  
    my $val = shift;  
    $val =~ s/\r?\n|\r\n?/\r/g;  
    return $val;  
},
```

2.28.2.10. name

Transposes a LAST, First name pair.

```
'name' => sub {  
    my $val = shift;  
    return $val unless $val =~ /,/;  
    my($last, $first) = split /\s*,\s*/, $val, 2;  
    return "$first $last";  
},
```

2.28.2.11. namecase

Namecases the text. Only works on values that are uppercase in the first letter, i.e. [filter op=namecase]LEONARDO da Vinci[/filter] will return "Leonardo da Vinci".

```
'namecase' => sub {  
    my $val = shift;  
    $val =~ s/([A-Z]\w+)/\L\u$1/g;  
    return $val;  
},
```

2.28.2.12. no_white

Strips all whitespace.

```
'no_white' => sub {  
    my $val = shift;  
    $val =~ s/\s+//g;  
    return $val;  
},
```

2.28.2.13. pagefile

Strips leading slashes and dots.

```
'pagefile' => sub {  
    $_[0] =~ s:^[./]+::;  
    return $_[0];  
},
```


2.28.2.14. sql

Change single–quote characters into doubled versions, i.e. ' becomes ".

```
'sql' => sub {
    my $val = shift;
    $val =~ s/':':g; # '
    return $val;
},
```

2.28.2.15. strip

Strips leading and trailing whitespace.

```
'strip' => sub {
    my $val = shift;
    $val =~ s/^\s+//;
    $val =~ s/\s+$//;
    return $val;
},
```

2.28.2.16. text2html

Rudimentary HTMLizing of text.

```
'text2html' => sub {
    my $val = shift;
    $val =~ s|\r?\n\r?\n|<P>|;
    $val =~ s|\r?\n|<BR>|;
    return $val;
},
```

2.28.2.17. uc

Uppercases the text.

```
'uc' => sub {
    return uc(shift);
},
```

2.28.2.18. unix

Removes those crufty carriage returns.

```
'unix' => sub {
    my $val = shift;
    $val =~ s/\r?\n/\n/g;
    return $val;
},
```

2.28.2.19. urlencode

Changes non–word characters (except colon) to %3c notation.

```
'urlencode' => sub {
    my $val = shift;
```

Interchange Tags Reference

```
$val =~ s|[^\\w:]|sprintf "%%02x", ord $1|eg;
return $val;
},
```

2.28.2.20. value

Returns the value of the user session variable. Useful for starting a filter sequence with a seed value.

```
'value' => sub {
    return $::Values->(shift);
},
```

2.28.2.21. word

Only returns word characters. Locale does apply if collation is properly set.

```
'word' => sub {
    my $val = shift;
    $val =~ s/\\W+//g;
    return $val;
},
```

You can define your own filters in an GlobalSub (or Sub or ActionMap):

```
package Vend::Interpolate;

$Filter{reverse} = sub { $val = shift; return scalar reverse $val };
```

That filter will reverse the characters sent.

The arguments sent to the subroutine are the value to be filtered, any associated variable or tag name, and any arguments appended to the filter name with periods as the separator.

A `[filter op=lookup.products.price]99-102[/filter]` will send ('99-102', undef, 'products', 'price') as the parameters. Assuming the value of the user variable `foo` is `bar`, the call `[value name=foo filter="lookup.products.price.extra"]` will send ('bar', 'foo', 'products', 'price', 'extra').

2.29. flag

2.29.1. Summary

Parameters: **type**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->flag(
    {
        type => VALUE,
    }
)
```

OR

```
$Tag->flag($type, $ATTRHASH);
```

Attribute aliases

```
flag ==> type
name ==> type
```

2.29.2. Description

NO Description

2.30. fly_list

2.30.1. Summary

Parameters: **code base**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [fly_list] FOO [/fly_list]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->fly_list(
    {
        code => VALUE,
        base => VALUE,
    },
    BODY
)
```

OR

```
$Tag->fly_list($code, $base, $BODY);
```

2.30.2. Description

Syntax: `[fly-list prefix=tag_prefix* code=code*]`

Defines an area in a random page which performs the flypage lookup function, implementing the tags below.

```
[fly-list]
  (contents of flypage.html)
[/fly-list]
```

If you place the above around the contents of the demo flypage, in a file named `flypage2.html`, it will make these two calls display identical pages:

```
[page 00-0011] One way to display the Mona Lisa [/page]
[page flypage2 00-0011] Another way to display the Mona Lisa [/page]
```

If you place a `[fly-list]` tag alone at the top of the page, it will cause any page to act as a flypage.

By default, the prefix is `item`, meaning the `[item-code]` tag will display the code of the item, the `[item-price]` tag will display price, etc. But if you use the prefix, i.e. `[fly-list prefix=fly]`, then it will be `[fly-code]`; `prefix=foo` would cause `[foo-code]`, etc.

2.31. fly_tax

2.31.1. Summary

Parameters: **area**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->fly_tax(
  {
    area => VALUE,
  }
)
```

OR

```
$Tag->fly_tax($area);
```

2.31.2. Description

Builds a tax rate from `taxarea`, `taxrate`, `taxshipping`, variable values, and the `SalesTax` directive value.

2.32. goto

2.32.1. Summary

Parameters: **name if**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->goto(
    {
        name => VALUE,
        if => VALUE,
    }
)
```

OR

```
$Tag->goto($name, $if);
```

2.32.2. Description

NO Description

2.33. handling

2.33.1. Summary

Parameters: **mode**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->handling(  
    {  
        mode => VALUE,  
    }  
)
```

OR

```
$Tag->handling($mode, $ATTRHASH);
```

Attribute aliases

```
cars ==> cart  
modes ==> mode  
name ==> mode  
tables ==> table
```

2.33.2. Description

Calculates and inserts handling costs. Accepts the same noformat and convert arguments as the shipping tag.

2.34. harness

2.34.1. Summary

No parameters.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [harness] FOO [/harness]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->harness(  
    {  
    },  
    BODY  
)
```

OR

```
$Tag->harness($ATTRHASH, $BODY);
```

2.34.2. Description

NO Description

2.35. href

Alias for [\[area\]](#) tag.

2.36. html_table

2.36.1. Summary

No parameters.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [html_table] FOO [/html_table]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->html_table(  
    {  
    },  
    BODY  
)
```

OR

```
$Tag->html_table($ATTRHASH, $BODY);
```

2.36.2. Description

NO Description

2.37. if

2.37.1. Summary

Parameters: **type term op compare**

THIS TAG HAS SPECIAL POSITIONAL PARAMETER HANDLING.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [if] FOO [/if]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

Called Routine for positional:

ASP/perl tag calls:

```
$Tag->if(  
  {  
    type => VALUE,  
    term => VALUE,  
    op => VALUE,  
    compare => VALUE,  
  },  
  BODY  
)
```

OR

```
$Tag->if($type, $term, $op, $compare, $BODY);
```

Attribute aliases

```
base ==> type  
comp ==> compare  
condition ==> compare  
operator ==> op
```

2.37.2. Description

Named call example: [if type="type" term="field" op="op" compare="compare"]

Positional call example: [if type field op compare]

negated: [if type="!type" term="field" op="op" compare="compare"]

Positional call example: [if !type field op compare]

Allows conditional building of HTML based on the setting of various Interchange session and database values. The general form is:

```
[if type term op compare]  
[then]
```

If true, this is printed on the document.
The [then] [/then] is optional in most

Interchange Tags Reference

cases. If ! is prepended to the type setting, the sense is reversed and this will be output for a false condition.

```
[/then]
[elsif type term op compare]
                                Optional, tested when if fails
[/elsif]
[else]
                                Optional, printed when all above fail
[/else]
[/if]
```

The [\[if\]](#) tag can also have some variants:

```
[if type=explicit compare=`$perl_code`]
    Displayed if valid Perl CODE returns a true value.
[/if]
```

You can do some Perl-style regular expressions:

```
[if value name =~ /^mike/]
                                This is the if with Mike.
[elsif value name =~ /^sally/]
                                This is an elsif with Sally.
[/elsif]
[elsif value name =~ /^pat/]
                                This is an elsif with Pat.
[/elsif]
[else]
                                This is the else, no name I know.
[/else]
[/if]
```

While named parameter tag syntax works for `[if ...]`, it is more convenient to use positional calls in most cases. The only exception is if you are planning on doing a test on the results of another tag sequence:

```
[if value name =~ /[value b_name]/]
    Shipping name matches billing name.
[/if]
```

Oops! This will not work. You must do instead

```
[if base=value field=name op="=~" compare="/[value b_name]/"]
    Shipping name matches billing name.
[/if]
```

or better yet

```
[if type=explicit compare=`
                                $Value->{name} =~ /$Value->{b_name}/
                                `]
    Shipping name matches billing name.
[/if]
```

Interchange also supports a limited `[and ...]` and `[or ...]` capability:

```
[if value name =~ /Mike/]
[or value name =~ /Jean/]
```

```
Your name is Mike or Jean.
[/if]

[if value name =~ /Mike/]
[and value state =~ /OH/]
Your name is Mike and you live in Ohio.
[/if]
```

If you wish to do very complex AND and OR operations, you will have to use `[if explicit]` or better yet embedded Perl/ASP. This allows complex testing and parsing of values.

There are many test targets available:

2.37.2.1. config Directive

The Interchange configuration variables. These are set by the directives in your Interchange configuration file (or the defaults).

```
[if config CreditCardAuto]
Auto credit card validation is enabled.
[/if]
```

2.37.2.2. data database::field::key

The Interchange databases. Retrieves a field in the database and returns true or false based on the value.

```
[if data products::size::99-102]
There is size information.
[else]
No size information.
[/else]
[/if]

[if data products::size::99-102 =~ /small/i]
There is a small size available.
[else]
No small size available.
[/else]
[/if]
```

2.37.2.3. discount

Checks to see if a discount is present for an item.

```
[if discount 99-102]
Item is discounted.
[/if]
```

2.37.2.4. explicit

A test for an explicit value. If perl code is placed between a `[condition]` `[/condition]` tag pair, it will be used to make the comparison. Arguments can be passed to import data from user space, just as with the `[perl]` tag.

```
[if explicit]
[condition]
    $country = '[value country]';
```

Interchange Tags Reference

```
        return 1 if $country =~ /u\.?s\.?a?/i;
        return 0;
[/condition]
You have indicated a US address.
[/else]
You have indicated a non-US address.
[/else]
[/if]
```

This example is a bit contrived, as the same thing could be accomplished with `[if value country =~ /u\.?s\.?a?/i]`, but you will run into many situations where it is useful.

This will work for *Variable* values:

```
[if type=explicit compare="__MYVAR__"] .. [/if]
```

2.37.2.5. file

Tests for existence of a file. Useful for placing image tags only if the image is present.

```
[if file /home/user/www/images/[item-code].gif]
<IMG SRC="[item-code].gif">
[/if]
```

The `file` test requires that the *SafeUntrap* directive contains `ftfile` (which is the default).

2.37.2.6. items

The Interchange shopping carts. If not specified, the cart used is the main cart. Usually used as a litmus test to see if anything is in the cart, for example:

```
[if items]You have items in your shopping cart.[/if]

[if items layaway]You have items on layaway.[/if]
```

2.37.2.7. ordered

Order status of individual items in the Interchange shopping carts. If not specified, the cart used is the main cart. The following items refer to a part number of 99–102.

```
[if ordered 99-102] Item 99-102 is in your cart. [/if]
  Checks the status of an item on order, true if item
  99-102 is in the main cart.

[if ordered 99-102 layaway] ... [/if]
  Checks the status of an item on order, true if item
  99-102 is in the layaway cart.

[if ordered 99-102 main size] ... [/if]
  Checks the status of an item on order in the main cart,
  true if it has a size attribute.

[if ordered 99-102 main size =~ /large/i] ... [/if]
  Checks the status of an item on order in the main cart,
  true if it has a size attribute containing 'large'.
```

To make sure it is exactly large, you could use:

```
[if ordered 99-102 main size eq 'large'] ... [/if]
```

2.37.2.8. scratch

The Interchange scratchpad variables, which can be set with the `[set name]value[/set]` element.

```
[if scratch mv_separate_items]
ordered items will be placed on a separate line.
[else]
ordered items will be placed on the same line.
[/else]
[/if]
```

2.37.2.9. session

the interchange session variables. of particular interest are `i<login>`, `i<frames>`, `i<secure>`, and `i<browser>`.

2.37.2.10. validcc

a special case, takes the form `[if validcc no type exp_date]`. evaluates to true if the supplied credit card number, type of card, and expiration date pass a validity test. does a luhn-10 calculation to weed out typos or phony card numbers. Uses the standard `CreditCardAuto` variables for targets if nothing else is passed.

2.37.2.11. value

the interchange user variables, typically set in search, control, or order forms. variables beginning with `c<mv_>` are interchange special values, and should be tested/used with caution.

The *field* term is the specifier for that area. For example, `[if session logged_in]` would return true if the `logged_in` session parameter was set.

As an example, consider buttonbars for frame-based setups. It would be nice to display a different buttonbar (with no frame targets) for sessions that are not using frames:

```
[if scratch frames]
__BUTTONBAR_FRAMES__
[else]
__BUTTONBAR__
[/else]
[/if]
```

Another example might be the when search matches are displayed. If you use the string `'[value mv_match_count] titles found'`, it will display a plural for only one match. Use:

```
[if value mv_match_count != 1]
[value mv_match_count] matches found.
[else]
Only one match was found.
[/else]
[/if]
```

The *op* term is the compare operation to be used. Compare operations are as in Perl:

```
== numeric equivalence
```

```
eq  string equivalence
>   numeric greater-than
gt  string greater-than
<   numeric less-than
lt  string less-than
!=  numeric non-equivalence
ne  string equivalence
```

Any simple perl test can be used, including some limited regex matching. More complex tests are best done with `[if explicit]`.

2.37.2.12. `[then] text [/then]`

This is optional if you are not nesting if conditions, as the text immediately following the `[if ..]` tag is used as the conditionally substituted text. If nesting `[if ...]` tags you should use a `[then][/then]` on any outside conditions to ensure proper interpolation.

2.37.2.13. `[elsif type field op* compare*]`

named attributes: `[elsif type="type" term="field" op="op" compare="compare"]`

Additional conditions for test, applied if the initial `[if ...]` test fails.

2.37.2.14. `[else] text [/else]`

The optional else-text for an if or if_field conditional.

2.37.2.15. `[condition] text [/condition]`

Only used with the `[if explicit]` tag. Allows an arbitrary expression **in Perl** to be placed inside, with its return value interpreted as the result of the test. If arguments are added to `[if explicit args]`, those will be passed as arguments are in the *[perl]* construct.

2.38. import

2.38.1. Summary

Parameters: **table type**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Interpolates **container text** by default>.

This is a container tag, i.e. `[import] FOO [/import]`. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->import(
    {
        table => VALUE,
        type => VALUE,
    },
    BODY
)
```

OR

```
$Tag->import($table, $type, $ATTRHASH, $BODY);
```

Attribute aliases

```
base ==> table
database ==> table
```

2.38.2. Description

Named attributes:

```
[import table=table_name
    type=(TAB|PIPE|CSV|%%|LINE)
    continue=(NOTES|UNIX|DITTO)
    separator=c]
```

Import one or more records into a database. The `type` is any of the valid Interchange delimiter types, with the default being defined by the setting of the database *DELIMITER*. The table must already be a defined Interchange database table; it cannot be created on the fly. (If you need that, it is time to use SQL.)

The `type` of `LINE` and `continue` setting of `NOTES` is particularly useful, for it allows you to name your fields and not have to remember the order in which they appear in the database. The following two imports are identical in effect:

```
[import table=orders]
code: [value mv_order_number]
shipping_mode: [shipping-description]
status: pending
[/import]

[import table=orders]
shipping_mode: [shipping-description]
status: pending
code: [value mv_order_number]
[/import]
```

The `code` or `key` must always be present, and is always named `code`.

If you do not use `NOTES` mode, you must import the fields in the same order as they appear in the ASCII source file.

The `[import] TEXT [/import]` region may contain multiple records. If using `NOTES` mode, you must use a separator, which by default is a form-feed character (^L).

2.39. include

2.39.1. Summary

Parameters: **file locale**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->include(
    {
        file => VALUE,
        locale => VALUE,
    }
)
```

OR

```
$Tag->include($file, $locale);
```

2.39.2. Description

Same as [file name] except interpolates for all Interchange tags and variables. Does NOT do locale translations.

2.40. index

2.40.1. Summary

Parameters: **table**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here.

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->index(  
  {  
    table => VALUE,  
  }  
)
```

OR

```
$Tag->index($table, $ATTRHASH);
```

Attribute aliases

```
base ==> table  
database ==> table
```

2.40.2. Description

NO Description

2.41. input_filter

2.41.1. Summary

Parameters: **name**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [input_filter] FOO [/input_filter]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->input_filter(  
  {  
    name => VALUE,  
  },  
  BODY  
)
```

OR

```
$Tag->input_filter($name, $ATTRHASH, $BODY);
```

Attribute aliases


```
ops ==> op
var ==> name
variable ==> name
```

2.41.2. Description

NO Description

2.42. item_list

2.42.1. Summary

Parameters: **name**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [item_list] FOO [/item_list]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->item_list(
    {
        name => VALUE,
    },
    BODY
)
```

OR

```
$Tag->item_list($name, $ATTRHASH, $BODY);
```

Attribute aliases

```
cart ==> name
```

2.42.2. Description

Within any page, the [item_list cart*] element shows a list of all the items ordered by the customer so far. It works by repeating the source between [item_list] and [/item_list] once for each item ordered.

NOTE: The special tags that reference item within the list are not normal Interchange tags, do not take named attributes, and cannot be contained in an HTML tag (other than to substitute for one of its values or provide a conditional container). They are interpreted only inside their corresponding list container. Normal

Interchange tags can be interspersed, though they will be interpreted *after* all of the list-specific tags.

Between the `item_list` markers the following elements will return information for the current item:

2.42.2.1. `[if-data table column]`

If the database field `column` in table *table* is non-blank, the following text up to the `[/if_data]` tag is substituted. This can be used to substitute `IMG` or other tags only if the corresponding source item is present. Also accepts a `[else]else text[/else]` pair for the opposite condition.

2.42.2.2. `[if-data ! table column]`

Reverses sense for `[if-data]`.

2.42.2.3. `[/if-data]`

Terminates an `[if_data table column]` element.

2.42.2.4. `[if-field fieldname]`

If the products database field *fieldname* is non-blank, the following text up to the `[/if_field]` tag is substituted. If you have more than one products database table (see *ProductFiles*), it will check them in order until a matching key is found. This can be used to substitute `IMG` or other tags only if the corresponding source item is present. Also accepts a `[else]else text[/else]` pair for the opposite condition.

2.42.2.5. `[if-field ! fieldname]`

Reverses sense for `[if-field]`.

2.42.2.6. `[/if-field]`

Terminates an `[if_field fieldname]` element.

2.42.2.7. `[item-accessories attribute*, type*, field*, database*, name*]`

Evaluates to the value of the Accessories database entry for the item. If passed any of the optional arguments, initiates special processing of item attributes based on entries in the product database.

2.42.2.8. `[item-code]`

Evaluates to the product code for the current item.

2.42.2.9. `[item-data database fieldname]`

Evaluates to the field name *fieldname* in the arbitrary database table *database*, for the current item.

2.42.2.10. `[item-description]`

Evaluates to the product description (from the products file) for the current item.

In support of OnFly, if the description field is not found in the database, the `description` setting in the

shopping cart will be used instead.

2.42.2.11. [item-field fieldname]

Evaluates to the field name *fieldname* in the products database, for the current item. If the item is not found in the first of the *ProductFiles*, all will be searched in sequence.

2.42.2.12. [item-increment]

Evaluates to the number of the item in the match list. Used for numbering search matches or order items in the list.

2.42.2.13. [item-last]tags[/item-last]

Evaluates the output of the Interchange tags encased inside the tags, and if it evaluates to a numerical non-zero number (i.e. 1, 23, or -1) then the list iteration will terminate. If the evaluated number is **negative**, then the item itself will be skipped. If the evaluated number is **positive**, then the item itself will be shown but will be last on the list.

```
[item-last][calc]
  return -1 if '[item-field weight]' eq '';
  return 1 if '[item-field weight]' < 1;
  return 0;
[/calc][/item-last]
```

If this is contained in your [\[item-list\]](#) (or [\[search-list\]](#) or flypage) and the weight field is empty, then a numerical -1 will be output from the [calc][calc] tags; the list will end and the item will **not** be shown. If the product's weight field is less than 1, a numerical 1 is output. The item will be shown, but will be the last item shown. (If it is an [\[item-list\]](#), any price for the item will still be added to the subtotal.)

NOTE: no HTML style.

2.42.2.14. [item-modifier attribute]

Evaluates to the modifier value of *attribute* for the current item.

2.42.2.15. [item-next]tags[/item_next]

Evaluates the output of the Interchange tags encased inside, and if it evaluates to a numerical non-zero number (i.e. 1, 23, or -1) then the item will be skipped with no output. Example:

```
[item-next][calc][item-field weight] < 1[/calc][/item-next]
```

If this is contained in your [\[item-list\]](#) (or [\[search-list\]](#) or flypage) and the product's weight field is less than 1, then a numerical 1 will be output from the [calc][calc] operation. The item will not be shown. (If it is an [\[item-list\]](#), any price for the item will still be added to the subtotal.)

2.42.2.16. [item-price n* noformat*]

Evaluates to the price for quantity *n* (from the products file) of the current item, with currency formatting. If the optional "noformat" is set, then currency formatting will not be applied.

2.42.2.17. [discount-price n* noformat*]

Evaluates to the discount price for quantity *n* (from the products file) of the current item, with currency formatting. If the optional "noformat" is set, then currency formatting will not be applied. Returns regular price if not discounted.

2.42.2.18. [item-discount]

Returns the difference between the regular price and the discounted price.

2.42.2.19. [item-quantity]

Evaluates to the quantity ordered for the current item.

2.42.2.20. [item-subtotal]

Evaluates to the subtotal (quantity * price) for the current item. Quantity price breaks are taken into account.

2.42.2.21. [modifier-name attribute]

Evaluates to the name to give an input box in which the customer can specify the modifier to the ordered item.

2.42.2.22. [quantity-name]

Evaluates to the name to give an input box in which the customer can enter the quantity to order.

2.43. label

2.43.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->label(  
  {  
    name => VALUE,  
  }  
)
```

OR

```
$Tag->label($name);
```

2.43.2. Description

NO Description

2.44. log

2.44.1. Summary

Parameters: **file**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. This may mean that there are parameters not shown here.

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [log] FOO [/log]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->log(
    {
        file => VALUE,
    },
    BODY
)
```

OR

```
$Tag->log($file, $ATTRHASH, $BODY);
```

Attribute aliases

```
arg ==> file
```

2.44.2. Description

NO Description

2.45. loop

2.45.1. Summary

Parameters: **list**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[loop] FOO [/loop]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->loop(
    {
        list => VALUE,
    },
    BODY
)
```

OR

```
$Tag->loop($list, $ATTRHASH, $BODY);
```

Attribute aliases

```
arg ==> list
args ==> list
```

2.45.2. Description

HTML example:

```
<TABLE><TR MV="loop 1 2 3"><TD>[loop-code]</TD></TR></TABLE>
```

Returns a string consisting of the LIST, repeated for every item in a comma-separated or space-separated list. Operates in the same fashion as the `[item-list]` tag, except for order-item-specific values. Intended to pull multiple attributes from an item modifier — but can be useful for other things, like building a pre-ordained product list on a page.

Loop lists can be nested reliably in Interchange by using the `prefix="tag"` parameter. New syntax:

```
[loop arg="A B C"]
    [loop prefix=mid arg="[loop-code]1 [loop-code]2 [loop-code]3"]
        [loop prefix=inner arg="X Y Z"]
            [mid-code]-[inner-code]
        [/loop]
    [/loop]
```

```
[ /loop]
```

You can do an arbitrary search with the `search="args"` parameter, just as in a one-click search:

```
[loop search="se=Americana/sf=category"]
  [loop-code] [loop-field title]
[/loop]
```

The above will show all items with a category containing the whole word "Americana", and will work the same in both old and new syntax.

2.45.2.1. **[if-loop-data table column] IF [else] ELSE [/else][if-loop-field]**

Outputs the **IF** if the `column` in `table` is non-empty, and the **ELSE** (if any) otherwise.

See [if-PREFIX-data].

2.45.2.2. **[if-loop-field column] IF [else] ELSE [/else][if-loop-field]**

Outputs the **IF** if the `column` in the `products` table is non-empty, and the **ELSE** (if any) otherwise. Will fall through to the first non-empty field if there are multiple `ProductFiles`.

See [if-PREFIX-field].

2.45.2.3. **[if-loop-param param] IF [else] ELSE [/else][if-loop-param]**

Only works if you have named return fields from a search (or from a passed list with the `lr=1` parameter).

Outputs the **IF** if the returned `param` is non-empty, and the **ELSE** (if any) otherwise.

See [if-PREFIX-param].

2.45.2.4. **[if-loop-pos N] IF [else] ELSE [/else][if-loop-param]**

Only works if you have multiple return fields from a search (or from a passed list with the `lr=1` parameter).

Parameters are numbered from ordinal 0, with `[loop-pos 0]` being the equivalent of `[loop-code]`.

Outputs the **IF** if the returned positional parameter `N` is non-empty, and the **ELSE** (if any) otherwise.

See [if-PREFIX-pos].

2.45.2.5. **[loop-accessories]**

Outputs an `[accessories ...]` item.

See [PREFIX-accessories].

2.45.2.6. [loop-change marker]

See [PREFIX-change].

2.45.2.7. [loop-code]

Evaluates to the code for the current item.

See [PREFIX-code].

2.45.2.8. [loop-data database fieldname]

Evaluates to the field name *fieldname* in the arbitrary database table *database*, for the current item.

See [PREFIX-data].

2.45.2.9. [loop-description]

Evaluates to the product description (from the products file, passed description in on-fly item, or description attribute in cart) for the current item.

See [PREFIX-description].

2.45.2.10. [loop-field fieldname]

Evaluates to the field name *fieldname* in the database, for the current item.

See [PREFIX-field].

2.45.2.11. [loop-increment]

Evaluates to the number of the item in the list. Used for numbering items in the list.

Starts from integer 1.

See [PREFIX-increment].

2.45.2.12. [loop-last]tags[/loop-last]

Evaluates the output of the Interchange tags encased inside, and if it evaluates to a numerical non-zero number (i.e. 1, 23, or -1) then the loop iteration will terminate. If the evaluated number is **negative**, then the item itself will be skipped. If the evaluated number is **positive**, then the item itself will be shown but will be last on the list.

```
[loop-last][calc]
  return -1 if '[loop-field weight]' eq '';
  return 1 if '[loop-field weight]' < 1;
  return 0;
[/calc][[/loop-last]
```

If this is contained in your `[loop list]` and the weight field is empty, then a numerical -1 will be output from the `[calc][[/calc]` tags; the list will end and the item will **not** be shown. If the product's weight field is less

than 1, a numerical 1 is output. The item will be shown, but will be the last item shown.

2.45.2.13. [loop-next]tags[/loop-next]

Evaluates the output of the Interchange tags encased inside, and if it evaluates to a numerical non-zero number (i.e. 1, 23, or -1) then the loop will be skipped with no output. Example:

```
[loop-next][calc][loop-field weight] < 1[/calc][[/loop-next]
```

If this is contained in your [loop list] and the product's weight field is less than 1, then a numerical 1 will be output from the [calc][calc] operation. The item will not be shown.

2.45.2.14. [loop-price n* noformat*]

Evaluates to the price for optional quantity n (from the products file) of the current item, with currency formatting. If the optional "noformat" is set, then currency formatting will not be applied.

2.46. mail

2.46.1. Summary

Parameters: **to**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [mail] FOO [/mail]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->mail(  
  {  
    to => VALUE,  
  },  
  BODY  
)
```

OR

```
$Tag->mail($to, $ATTRHASH, $BODY);
```

2.46.2. Description

NO Description

2.47. mvasp

2.47.1. Summary

Parameters: **tables**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [mvasp] FOO [/mvasp]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->mvasp(  
    {  
        tables => VALUE,  
    },  
    BODY  
)
```

OR

```
$Tag->mvasp($tables, $ATTRHASH, $BODY);
```

Attribute aliases

```
table ==> tables
```

2.47.2. Description

NO Description

2.48. nitems

2.48.1. Summary

Parameters: **name**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->nitems(  
    {  
        name => VALUE,  
    }  
)
```

OR

```
$Tag->nitems($name, $ATTRHASH);
```

2.48.2. Description

Expands into the total number of items ordered so far. Takes an optional cart name as a parameter.

2.49. onfly

2.49.1. Summary

Parameters: **code quantity**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->onfly(  
    {  
        code => VALUE,  
        quantity => VALUE,  
    }  
)
```

OR

```
$Tag->onfly($code, $quantity, $ATTRHASH);
```

2.49.2. Description

NO Description

2.50. options

2.50.1. Summary

Parameters: **code**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->options(  
  {  
    code => VALUE,  
  }  
)
```

OR

```
$Tag->options($code, $ATTRHASH);
```

2.50.2. Description

NO Description

2.51. or

2.51.1. Summary

Parameters: **type term op compare**

THIS TAG HAS SPECIAL POSITIONAL PARAMETER HANDLING.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

Called Routine for positional:

ASP/perl tag calls:

```
$Tag->or(
    {
        type => VALUE,
        term => VALUE,
        op => VALUE,
        compare => VALUE,
    }
)
```

OR

```
$Tag->or($type, $term, $op, $compare);
```

Attribute aliases

```
base ==> type
comp ==> compare
operator ==> op
```

2.51.2. Description

NO Description

2.52. order

2.52.1. Summary

Parameters: **code quantity**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->order(
    {
        code => VALUE,
        quantity => VALUE,
    }
)
```

```
}
)
```

OR

```
$Tag->order($code, $quantity);
```

2.52.2. Description

Expands into a hypertext link which will include the specified code in the list of products to order and display the order page. **code** should be a product code listed in one of the "products" databases. The optional argument **cart/page** selects the shopping cart the item will be placed in (begin with / to use the default cart main) and the order page that will display the order. The optional argument **database** constrains the order to a particular products file — if not specified, all databases defined as products files will be searched in sequence for the item.

Example:

```
Order a [order TK112]Toaster[/order] today.
```

Note: [/order] simply expands to .

2.53. page

Expands to a hyperlink to an Interchange page or action, including surrounding <A HREF ...>. The URL within the link includes the Interchange session ID and supplied arguments. The optional [/page] is simply a macro for .

If you do not want the <A HREF ...>, use the [area](#) tag instead — these are equivalent:

```
[page href=dir/page arg=mv_arg]TargetName[/page]
<A HREF="[area href=dir/page arg=mv_arg]">TargetName</A>
```

2.53.1. Summary

```
[page href arg other named attributes]
[page href=dir/page arg=page_arguments other named attributes]
```

<i>Parameters</i>	<i>Description</i>	<i>Default</i>
href	Path to Interchange page or action <i>Special arguments</i> <ul style="list-style-type: none"> ◆ 'scan' treats arg as a search argument ◆ 'http://...' external link (requires form attribute) 	process
arg	Interchange arguments to page or action	none
base	alias for arg	none

<i>Attributes</i>	<i>Default</i>
-------------------	----------------

extra	<i>none</i>
form	<i>none</i>
search	<i>No</i>
secure	<i>No</i>
interpolate (reparse)	<i>No</i>

<i>Other_Characteristics</i>	
Invalidates cache	<i>No</i>
Macro	<i>No</i>
Has end tag	<i>No</i> ([/page] is a macro for)

Tag expansion example:

```
[page href=dir/page.html arg="arg1=AA/arg2=BB" ]

<a href="www.here.com/cgi-bin/mycatalog/page.html?mv_session_id=6CZ2whqo&\
mv_pc=1&mv_arg=arg1%3dAA/arg2%3dBB">
```

ASP-like Perl call:

```
$Tag->page( { href => "dir/page",
              arg  => "arguments", } );
```

or similarly with positional parameters,

```
$Tag->page($href, $arg, $attribute_hash_reference);
```

2.53.1.1. See Also

[area](#)

2.53.2. Description

The page tag inserts a hyperlink to the specified Interchange page or action. For example, [page shirts] will expand into

```
<a href="http://www.here.com/cgi-bin/mycatalog/shirts?mv_session_id=6CZ2whqo&mv_pc=1">
```

The catalog page displayed will come from "shirts.html" in the pages directory.

The additional argument will be passed to Interchange and placed in the {arg} session parameter. This allows programming of a conditional page display based on where the link came from. The argument is then available with the tag [data session arg], or the embedded Perl session variable \$Session->{arg}. Spaces and some other characters will be escaped with the %NN HTTP-style notation and unescaped when the argument is read back into the session.

For better performance, Interchange can prebuild and cache pages that would otherwise be generated dynamically. If Interchange has built such a static page for the target, the page tag produces a link to the

cached page whenever the user has accepted and sent back a cookie with the session ID. If the user did not accept the cookie, Interchange cannot use the cache, since the link must then include the *mv_session_id* argument in order to preserve session.

2.53.2.1. form

The optional *form* argument allows you to encode a form in the link.

```
[page form="mv_order_item=os28044
mv_order_size=15oz
mv_order_quantity=1
mv_separate_items=1
mv_todo=refresh"] Order 15oz Framing Hammer</A>
```

The two form values *mv_session_id* and *mv_arg* are automatically added when appropriate. The form value *mv_arg* receives the value of the tag's *arg* parameter.

This would generate a form that ordered quantity one of item number *os28044* with size *15oz*. The item would appear on a separate line in the shopping cart, since *mv_separate_items* is set. Since the *href* is not set, you will go to the default shopping cart page — alternatively, you could have set *mv_orderpage=yourpage* to go to *yourpage*.

All normal Interchange form caveats apply — you must have an action, you must supply a page if you don't want to go to the default, etc.

You can theoretically submit any form with this, though none of the included values can have newlines or trailing whitespace. If you want to do something like that you will have to write a UserTag.

If the parameter *href* is not supplied, *process* is used, causing normal Interchange form processing.

If the *href* points to an *http://* link, then no Interchange URL processing will be done, but the URL will include *mv_session_id*, *mv_pc*, and any arguments supplied with the *arg* attribute:

```
[page href="http://www.elsewhere.net/cgi/script"
form="cgi_1=ONE
cgi_2=TWO"
arg="interchange argument"]External link</A>

<A HREF="http://www.elsewhere.net/cgi/script?\
mv_session_id=6CZ2whqo&mv_pc=1&mv_arg=interchange%20argument&\
cgi_1=ONE&cgi_2=TWO">External link</A>
```

2.53.2.2. search

Interchange allows you to pass a search in a URL. There are two ways to do this:

1. Place the search specification in the named *search* attribute.
 - ◆ Interchange will ignore the *href* parameter (the link will be set to 'scan').
 - ◆ If you give the *arg* parameter a value, that value will be available as [[value](#) *mv_arg*] within the search display page.
2. Set the *href* parameter to 'scan' and set *arg* to the search specification.
 - ◆ Note that you can use this form positionally — the values go into *href* and *arg*, so you do not have to name parameters.

These are identical:

```
[page scan
    se=Impressionists
    sf=category]
    Impressionist Paintings
[/page]

[page href=scan
    arg="se=Impressionists
        sf=category"]
    Impressionist Paintings
</A>

[page search="se=Impressionists
    sf=category"]
    Impressionist Paintings
[/page]
```

Here is the same thing from a non-interchange page (e.g., a home page), assuming '/cgi-bin/mycatalog' is the CGI path to Interchange's vlink):

```
<A HREF="/cgi-bin/mycatalog/scan/se=Impressionists/sf=category">
    Impressionist Paintings
</A>
```

Sometimes, you will find that you need to pass characters that will not be interpreted positionally. In that case, you should quote the arguments:

```
[page href=scan
    arg=|
        se="Something with spaces"
    |]
```

The two-letter abbreviations are mapped with these letters:

DL	mv_raw_dict_look
MM	mv_more_matches
SE	mv_raw_searchspec
ac	mv_all_chars
ar	mv_arg
bd	mv_base_directory
bs	mv_begin_string
ck	mv_cache_key
co	mv_coordinate
cs	mv_case
cv	mv_verbatim_columns
de	mv_dict_end
df	mv_dict_fold
di	mv_dict_limit
dl	mv_dict_look
do	mv_dict_order
dp	mv_delay_page
dr	mv_record_delim
em	mv_exact_match
er	mv_spelling_errors
fi	mv_search_file
fm	mv_first_match

Interchange Tags Reference

```
fn mv_field_names
hs mv_head_skip
id mv_session_id
il mv_index_delim
ix mv_index_delim
lb mv_search_label
lo mv_list_only
lr mv_line_return
lr mv_search_line_return
ml mv_matchlimit
mm mv_max_matches
mp mv_profile
ms mv_min_string
ne mv_negate
np mv_nextpage
nu mv_numeric
op mv_column_op
os mv_orsearch
pc mv_pc
ra mv_return_all
rd mv_return_delim
rf mv_return_fields
rg mv_range_alpha
rl mv_range_look
rm mv_range_min
rn mv_return_file_name
rr mv_return_reference
rs mv_return_spec
rx mv_range_max
se mv_searchspec
sf mv_search_field
si mv_search_immediate
sp mv_search_page
sq mv_sql_query
st mv_searchtype
su mv_substring_match
tf mv_sort_field
to mv_sort_option
un mv_unique
va mv_value
```

They can be treated just the same as form variables on the page, except that they can't contain spaces, '/' in a file name, or quote marks. These characters can be used in URL hex encoding, i.e. %20 is a space, %2F is a /, etc. -- &sp; or will not be recognized. If you use one of the methods below to escape these "unsafe" characters, you won't have to worry about this.

You may specify a one-click search in three different ways. The first is as used in previous versions, with the scan URL being specified completely as the page name. The second two use the "argument" parameter to the [page . . .] or [[area](#) ...]> tags to specify the search (an argument to a scan is never valid anyway).

2.53.2.3. Original syntax

If you wish to do an OR search on the fields category and artist for the strings "Surreal" and "Gogh", while matching substrings, you would do:

```
[page scan se=Surreal/se=Gogh/os=yes/su=yes/sf=artist/sf=category]
  Van Gogh -- compare to surrealists
[/page]
```

In this method of specification, to replace a / (slash) in a file name (for the sp, bd, or fi parameter) you must use the shorthand of ::, i.e. sp=results::standard. (This may not work for some browsers, so you should probably either put the page in the main pages directory or define the page in a search profile.)

2.53.2.4. Ampersand syntax

You can substitute & for / in the specification and be able to use / and quotes and spaces in the specification.

```
[page scan se="Van Gogh"&sp=lists/surreal&os=yes&su=yes&sf=artist&sf=category]
  Van Gogh -- compare to surrealists
[/page]
```

Any "unsafe" characters will be escaped.

2.53.2.5. Multi-line syntax

You can specify parameters one to a line, as well.

```
[page scan
  se="Van Gogh"
  sp=lists/surreal
  os=yes
  su=yes
  sf=artist
  sf=category
] Van Gogh -- compare to surrealists [/page]
```

Any "unsafe" characters will be escaped. You may not search for trailing spaces in this method; it is allowed in the other notations.

2.53.2.6. [/page]

This is not an actual end tag, but simply a macro that expands to . Use it with the page element as follows:

```
[page shirts]Our shirt collection[/page].
```

TIP: A small efficiency boost in large pages is to just use the tag.

2.54. perl

2.54.1. Summary

Parameters: **tables**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. This may mean that there are parameters not shown here.

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [perl] FOO [/perl]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->perl(
    {
        tables => VALUE,
    },
    BODY
)
```

OR

```
$Tag->perl($tables, $ATTRHASH, $BODY);
```

Attribute aliases

```
table ==> tables
```

2.54.2. Description

```
[perl]
    $name    = $Values->{name};
    $browser = $Session->{browser};
    return "Hi, $name! How do you like your $browser?
[/perl]
```

HTML example:

```
<PRE mv=perl>
    $name    = $Values->{name};
    $browser = $Session->{browser};
    return "Hi, $name! How do you like your $browser?
</PRE>
```

Perl code can be directly embedded in Interchange pages. The code is specified as [perl arguments*] any_legal_perl_code [/perl]. The value returned by the code will be inserted on the page.

Object references are available for most Interchange tags and functions, as well as direct references to Interchange session and configuration values.

<code>\$CGI->{key}</code>	Hash reference to raw submitted values
<code>\$CGI_array->{key}</code>	Arrays of submitted values
<code>\$Carts->{cartname}</code>	Direct reference to shopping carts
<code>\$Config->{key}</code>	Direct reference to <code>\$Vend::Cfg</code>
<code>\$DbSearch->array(@args)</code>	Do a DB search and get results
<code>\$Document->header()</code>	Writes header lines
<code>\$Document->send()</code>	Writes to output
<code>\$Document->write()</code>	Writes to page
<code>\$Scratch->{key}</code>	Direct reference to scratch area
<code>\$Session->{key}</code>	Direct reference to session area
<code>\$Tag->tagname(@args)</code>	Call a tag as a routine (UserTag too!)

Interchange Tags Reference

<code>\$TextSearch->array(@args)</code>	Do a text search and get results
<code>\$Values->{key}</code>	Direct reference to user form values
<code>\$Variable->{key}</code>	Config variables (same as <code>\$Config->{Variable}</code>);
<code>&HTML(\$html)</code>	Same as <code>\$Document->write(\$html)</code> ;
<code>&Log(\$msg)</code>	Log to the error log

For full descriptions of these objects, see *Interchange Programming*.

If you wish to use database values in your Perl code, you must pre-open the table(s) you will be using. This can be done by including the table name in the `tables` parameter of the Perl tag:

```
[perl tables=products]
    $result = "You asked about $Values->{code}. Here is the description: ";
    $result .= $Tag->data('products', 'description', $Values->{code});
    return $result;
[/perl]
```

If you do not do this, your code will fail with a runtime Safe error.

2.55. price

2.55.1. Summary

Parameters: **code**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->price(
    {
        code => VALUE,
    }
)
```

OR

```
$Tag->price($code, $ATTRHASH);
```

Attribute aliases

```
base ==> mv_ib
```

2.55.2. Description

Arguments:

code	Product code/SKU
base	Only search in product table <i>*base*</i>
quantity	Price for a quantity
discount	If true(1), check discount coupons and apply
noformat	If true(1), don't apply currency formatting

Expands into the price of the product identified by code as found in the products database. If there is more than one products file defined, they will be searched in order unless constrained by the optional argument **base**. The optional argument **quantity** selects an entry from the quantity price list. To receive a raw number, with no currency formatting, use the option `noformat=1`.

Interchange maintains a price in its database for every product. The price field is the one required field in the product database — it is necessary to build the price routines.

For speed, Interchange builds the code that is used to determine a product's price at catalog configuration time. If you choose to change a directive that affects product pricing you must reconfigure the catalog.

Quantity price breaks are configured by means of the *CommonAdjust* directive. There are a number of CommonAdjust recipes which can be used; the standard example in the demo calls for a separate pricing table called `pricing`. Observe the following:

```
CommonAdjust pricing:q2,q5,q10,q25, ;products:price, ==size:pricing
```

This says to check quantity and find the applicable column in the pricing database and apply it. In this case, it would be:

2-4	Column <i>*q2*</i>
5-9	Column <i>*q5*</i>
10-24	Column <i>*q10*</i>
25 up	Column <i>*q25*</i>

What happens if quantity is one? It "falls back" to the price that is in the table `products`, column `price`.

After that, if there is a size attribute for the product, the column in the pricing database corresponding to that column is checked for additions or subtractions (or even percentage changes).

If you use this tag in the demo:

```
[price code=99-102 quantity=10 size=XL]
```

the price will be according to the `q10` column, adjusted by what is in the `XL` column. (The row is of course 99–102.) The following entry in `pricing`:

code	q2	q5	q10	q25	XL
99-102	10	9	8	7	.50

Would yield 8.50 for the price. Quantity of 10 in the `q10` column, with 50 cents added for extra large (XL).

Following are several examples based on the above entry as well as this the entry in the `products` table:

Interchange Tags Reference

code	description	price	size
99-102	T-Shirt	10.00	S=Small, M=Medium, L=Large*, XL=Extra Large

NOTE: The examples below assume a US locale with 2 decimal places, use of commas to separate, and a dollar sign (\$) as the currency formatting.

TAG	DISPLAYS
[price 99-102]	\$10.00
[price code="99-102"]	\$10.00
[price code="99-102" quantity=1]	\$10.00
[price code="99-102" noformat=1]	10
[price code="99-102" quantity=5]	\$9.00
[price code="99-102" quantity=5 size=XL]	\$9.50
[price code="99-102" size=XL]	\$10.50
[price code="99-102" size=XL noformat=1]	10.5

Product discounts for specific products, all products, or the entire order can be configured with the [discount ...] tag. Discounts are applied on a per-user basis -- you can gate the discount based on membership in a club or other arbitrary means.

Adding [discount 99-102] \$s * .9[/discount] deducts 10% from the price at checkout, but the price tag will not show that unless you add the discount=1 parameter.

[price code="99-102"]	-->	\$10.00
[price code="99-102" discount=1]	-->	\$9.00

See *Product Discounts*.

2.56. process

2.56.1. Summary

Aliases for tag

process_target

Parameters: **target secure**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->process(
```

```
{
  target => VALUE,
  secure => VALUE,
}
```

OR

```
$Tag->process($target, $secure, $ATTRHASH);
```

2.56.2. Description

NO Description

2.57. profile

2.57.1. Summary

Parameters: **name**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->profile(
  {
    name => VALUE,
  }
)
```

OR

```
$Tag->profile($name, $ATTRHASH);
```

2.57.2. Description

NO Description

2.58. query

2.58.1. Summary

Parameters: **sql**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[query] FOO [/query]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->query(
    {
        sql => VALUE,
    },
    BODY
)
```

OR

```
$Tag->query($sql, $ATTRHASH, $BODY);
```

Attribute aliases

```
query ==> sql
```

2.58.2. Description

NO Description

2.59. read_cookie

2.59.1. Summary

Parameters: **name**

ONLY THE PARAMETERS ARE POSITIONAL.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->read_cookie(  
    {  
        name => VALUE,  
    }  
)
```

OR

```
$Tag->read_cookie($name);
```

2.59.2. Description

NO Description

2.60. record

2.60.1. Summary

No parameters.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->record(  
    {  
    }  
)
```

OR

```
$Tag->record($ATTRHASH);
```

Attribute aliases

```
code ==> key  
column ==> col  
field ==> col
```

2.60.2. Description

NO Description

2.61. region

2.61.1. Summary

No parameters.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [region] FOO [/region]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->region(
    {
    },
    BODY
)
```

OR

```
$Tag->region($ATTRHASH, $BODY);
```

Attribute aliases

```
args ==> arg
params ==> arg
search ==> arg
```

2.61.2. Description

NO Description

2.62. row

2.62.1. Summary

Parameters: **width**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [row] FOO [/row]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->row(
  {
    width => VALUE,
  },
  BODY
)
```

OR

```
$Tag->row($width, $BODY);
```

2.62.2. Description

Formats text in tables. Intended for use in emailed reports or <PRE></PRE> HTML areas. The parameter *nn* gives the number of columns to use. Inside the row tag, [col param=value ...] tags may be used.

2.62.2.1. [col width=nn wrap=yes|no gutter=n align=left|right|input spacing=n]

Sets up a column for use in a [row]. This parameter can only be contained inside a [row nn] [/row] tag pair. Any number of columns (that fit within the size of the row) can be defined.

The parameters are:

width=nn	The column width, I<including the gutter>. Must be supplied, there is no default. A shorthand method is to just supply the number as the I<first> parameter, as in [col 20].
gutter=n	The number of spaces used to separate the column (on the right-hand side) from the next. Default is 2.
spacing=n	The line spacing used for wrapped text. Default is 1, or single-spaced.
wrap=(yes no)	Determines whether text that is greater in length than the column width will be wrapped to the next line. Default is I<yes>.
align=(L R I)	Determines whether text is aligned to the left (the default), the right, or in a way that might display an HTML text input field correctly.

2.62.2.2. [/col]

Terminates the column field.

2.63. salestax**2.63.1. Summary**

Parameters: **name noformat**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->salestax(
    {
        name => VALUE,
        noformat => VALUE,
    }
)
```

OR

```
$Tag->salestax($name, $noformat);
```

Attribute aliases

```
cart ==> name
```

2.63.2. Description

Expands into the sales tax on the subtotal of all the items ordered so far for the cart, default cart is main. If there is no key field to derive the proper percentage, such as state or zip code, it is set to 0. If the noformat tag is present and non-zero, the raw number with no currency formatting will be given.

2.64. scratch**2.64.1. Summary**

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->scratch(  
    {  
        name => VALUE,  
    }  
)
```

OR

```
$Tag->scratch($name);
```

2.64.2. Description

Returns the contents of a scratch variable to the page. (A scratch variable is set with a [set] value [/set] container pair.)

2.65. scratchd

2.65.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->scratchd(  
    {  
        name => VALUE,  
    }  
)
```

OR

```
$Tag->scratchd($name);
```

2.65.2. Description

NO Description

2.66. search_list

Formats results returned by a search. Must be enclosed within a [search_region](#). Has sub-tags (see [Looping](#) tags and Sub-tags).

2.67. search_region

2.67.1. Summary

Parameters: **arg**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [search_region] FOO [/search_region]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->search_region(
    {
        arg => VALUE,
    },
    BODY
)
```

OR

```
$Tag->search_region($arg, $ATTRHASH, $BODY);
```

Attribute aliases

```
args ==> arg
params ==> arg
search ==> arg
```

2.67.2. Description

NO Description

2.68. selected

2.68.1. Summary

Parameters: **name value**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here.

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->selected(
    {
        name => VALUE,
        value => VALUE,
    }
)
```

OR

```
$Tag->selected($name, $value, $ATTRHASH);
```

2.68.2. Description

You can provide a "memory" for drop-down menus, radio buttons, and checkboxes with the [checked] and [selected] tags.

This will output **SELECTED** if the variable `var_name` is equal to `value`. If the optional **MULTIPLE** argument is present, it will look for any of a variety of values. Not case sensitive unless the optional `case=1` parameter is used.

Here is a drop-down menu that remembers an item-modifier color selection:

```
<SELECT NAME="color">
<OPTION [selected color blue]> Blue
<OPTION [selected color green]> Green
<OPTION [selected color red]> Red
</SELECT>
```

Here is the same thing, but for a shopping-basket color selection

```
<SELECT NAME="[modifier-name color]">
<OPTION [selected [modifier-name color] blue]> Blue
<OPTION [selected [modifier-name color] green]> Green
<OPTION [selected [modifier-name color] red]> Red
```



```
</SELECT>
```

By default, the Values space (i.e. [value foo]) is checked — if you want to use the volatile CGI space (i.e. [cgi foo]) use the option `cgi=1`.

2.69. set

2.69.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. [set] FOO [/set]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->set(
  {
    name => VALUE,
  },
  BODY
)
```

OR

```
$Tag->set($name, $BODY);
```

2.69.2. Description

Sets a scratch variable to *value*.

Most of the `mv_*` variables that are used for search and order conditionals are in another namespace — they can be set by means of hidden fields in a form.

You can set an order profile with:

```
[set checkout]
name=required
address=required
[/set]
<INPUT TYPE=hidden NAME=mv_order_profile VALUE="checkout">
```

A search profile would be set with:

```
[set substring_case]
mv_substring_match=yes
mv_case=yes
[/set]
<INPUT TYPE=hidden NAME=mv_profile VALUE="substring_case">
```

Any of these profile values can be set in the OrderProfile files as well.

2.70. set_cookie

2.70.1. Summary

Parameters: **name value expire**

ONLY THE PARAMETERS ARE POSITIONAL.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->set_cookie(
{
    name => VALUE,
    value => VALUE,
    expire => VALUE,
}
```

```
)
```

OR

```
$Tag->set_cookie($name, $value, $expire);
```

2.70.2. Description

NO Description

2.71. seti

2.71.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [seti] FOO [/seti]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->seti(  
    {  
        name => VALUE,  
    },  
    BODY  
)
```

OR

```
$Tag->seti($name, $BODY);
```

2.71.2. Description

NO Description

2.72. setlocale

2.72.1. Summary

Parameters: **locale currency**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->setlocale(  
    {  
        locale => VALUE,  
        currency => VALUE,  
    }  
)
```

OR

```
$Tag->setlocale($locale, $currency, $ATTRHASH);
```

2.72.2. Description

NO Description

2.73. shipping

2.73.1. Summary

Parameters: **mode**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->shipping(
{
  mode => VALUE,
}
)
```

OR

```
$Tag->shipping($mode, $ATTRHASH);
```

Attribute aliases

```
cars => cart
modes => mode
name => mode
tables => table
```

2.73.2. Description

The shipping cost of the items in the basket via mode — the default mode is the shipping mode currently selected in the mv_shipmode variable. See *SHIPPING*.

2.74. shipping_desc

2.74.1. Summary

Aliases for tag

shipping_description

Parameters: **mode**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->shipping_desc(
    {
        mode => VALUE,
    }
)
```

OR

```
$Tag->shipping_desc($mode);
```

2.74.2. Description

NO Description

2.75. soap

2.75.1. Summary

Parameters: **call uri proxy**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->soap(
  {
    call => VALUE,
    uri  => VALUE,
    proxy => VALUE,
  }
)
```

OR

```
$Tag->soap($call, $uri, $proxy, $ATTRHASH);
```

2.75.2. Description

NO Description

2.76. sql

2.76.1. Summary

Parameters: **type query**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[sql] FOO [/sql]`. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->sql(
  {
    type => VALUE,
    query => VALUE,
  },
  BODY
)
```

OR

```
$Tag->sql($type, $query, $ATTRHASH, $BODY);
```

2.76.2. Description

NO Description

2.77. strip

2.77.1. Summary

No parameters.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [strip] FOO [/strip]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->strip(
    {
    },
    BODY
)
```

OR

```
$Tag->strip($BODY);
```

2.77.2. Description

NO Description

2.78. subtotal

2.78.1. Summary

Parameters: **name noformat**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->subtotal(  
    {  
        name => VALUE,  
        noformat => VALUE,  
    }  
)
```

OR

```
$Tag->subtotal($name, $noformat);
```

Attribute aliases

```
cart ==> name
```

2.78.2. Description

Positional: [subtotal cart* noformat*]

mandatory: NONE

optional: cart noformat

Expands into the subtotal cost, exclusive of sales tax, of all the items ordered so far for the optional `cart`. If the `noformat` tag is present and non-zero, the raw number with no currency formatting will be given.

2.79. tag

2.79.1. Summary

Parameters: **op arg**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. This may mean that there are parameters not shown here.

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. [tag] FOO [/tag]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->tag(  
    {  
        name => VALUE,  
        noformat => VALUE,  
    }  
)
```



```

    {
      op => VALUE,
      arg => VALUE,
    },
    BODY
  )

```

OR

```
$Tag->tag($op, $arg, $ATTRHASH, $BODY);
```

Attribute aliases

```
description ==> arg
```

2.79.2. Description

Performs any of a number of operations, based on the presence of `arg`. The arguments that may be given are:

2.79.2.1. export database file* type*

Exports a complete Interchange database to its text source file (or any specified file). The integer `n`, if specified, will select export in one of the enumerated Interchange export formats. The following tag will export the products database to `products.txt` (or whatever you have defined its source file as), in the format specified by the *Database* directive:

```
[tag export products][tag]
```

Same thing, except to the file `products/new_products.txt`:

```
[tag export products products/newproducts.txt][tag]
```

Same thing, except the export is done with a PIPE delimiter:

```
[tag export products products/newproducts.txt 5][tag]
```

The file is relative to the catalog directory, and only may be an absolute path name if *NoAbsolute* is set to No.

2.79.2.2. flag arg

Sets an Interchange condition.

The following enables writes on the `products` and `sizes` databases held in Interchange internal DBM format:

```
[tag flag write]products sizes[tag]
```

SQL databases are always writable if allowed by the SQL database itself — in-memory databases will never be written.

The `[tag flag build][tag]` combination forces static build of a page, even if dynamic elements are contained. Similarly, the `[tag flag cache][tag]` forces search or page caching (not usually wise).

2.79.2.3. log dir/file

Logs a message to a file, fully interpolated for Interchange tags. The following tag will send every item code and description in the user's shopping cart to the file logs/transactions.txt:

```
[tag log logs/transactions.txt]
[item_list][item-code] [item-description]
[/item_list][tag]
```

The file is relative to the catalog directory, and only may be an absolute path name if *NoAbsolute* is set to No.

2.79.2.4. mime description_string

Returns a MIME-encapsulated message with the boundary as employed in the other mime tags, and the *description_string* used as the Content-Description. For example

```
[tag mime My Plain Text]Your message here.[tag]
```

will return

```
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-ID: [sequential, lead as in mime boundary]
Content-Description: My Plain Text
```

```
Your message here.
```

When used in concert with [tag mime boundary], [tag mime header], and [tag mime id], allows MIME attachments to be included — typically with PGP-encrypted credit card numbers. See the demo page [ord/report.html](#) for an example.

2.79.2.5. mime boundary

Returns a MIME message boundary with unique string keyed on session ID, page count, and time.

2.79.2.6. mime header

Returns a MIME message header with the proper boundary for that session ID, page count, and time.

2.79.2.7. mime id

Returns a MIME message id with the proper boundary for that session ID, page count, and time.

2.79.2.8. show_tags

The encased text will not be substituted for with Interchange tags, with < and [characters changed to &#lt; and [respectively.

```
[tag show_tags][value whatever][tag]
```

2.79.2.9. time

Formats the current time according to POSIX strftime arguments. The following is the string for Thursday, April 30, 1997.

```
[tag time]%A, %B %d, %Y[/tag]
```

2.79.2.10. touch

Touches a database to allow use of the `tag_data()` routine in user-defined subroutines. If this is not done, the routine will error out if the database has not previously been accessed on the page.

```
[tag touch products][/tag]
```

2.80. time

2.80.1. Summary

Parameters: **locale**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[time] FOO [/time]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->time(
    {
        locale => VALUE,
    },
    BODY
)
```

OR

```
$Tag->time($locale, $ATTRHASH, $BODY);
```

2.80.2. Description

Formats the current time according to POSIX `strftime` arguments. The following is the string for Monday, January 1, 2001.

```
[time]%A, %B %d, %Y[/tag]
```

See the `strftime` man page for information on the arguments (which are the same as modern UNIX date commands).

Accepts the following options:

2.80.2.1. adjust

If you wish to temporarily adjust the time for display purposes, you can pass an `adjust` parameter with the number of hours (plus or minus) from the local time or from GMT:

```
[time]%c[/time]
[time adjust="-3"]%c[/time]
```

Will display:

```
Mon 01 Jan 2001 11:29:03 AM EST
Mon 01 Jan 2001 08:29:03 AM EST
```

Note that the time zone does not change — you should either pick a format which doesn't display zone, use the `tz` parameter, or manage it yourself.

NOTE: You can adjust time globally for an Interchange installation by setting the `$ENV{TZ}` variable on many systems. Set `TZ` in your environment and then restart interchange:

```
## bash/ksh/sh
TZ=PST7PDT; export TZ
interchange -restart

## csh/tcsh
setenv TZ PST7PDT
interchange -restart
```

On most modern UNIX systems, all times will now be in the PST zone.

2.80.2.2. gmt

If you want to display time as GMT, use the `gmt` parameter:

```
[time]%c[/time]
[time gmt=1]%c[/time]
```

will display:

```
Mon 01 Jan 2001 11:33:26 AM EST
Mon 01 Jan 2001 04:33:26 PM EST
```

Once again, the zone will not be set to GMT, so you should pick a format string which doesn't use zone, use the `tz` parameter, or manage it yourself.

2.80.2.3. locale

Format the time according to the named `locale`, assuming that locale is available on your operating system. For example, the following:

```
[time locale=en_US]%B %d, %Y[/time]
[time locale=fr_FR]%B %d, %Y[/time]
```

should display:

```
January 01, 2001  
janvier 01, 2001
```

2.80.2.4. tz

Use the passed `tz` to display the time. Will adjust for hours difference.

Example:

```
[time tz=GMT0]  
[time tz=CST6CDT]  
[time tz=PST8PDT]
```

will display:

```
Mon 01 Jan 2001 04:43:02 PM GMT  
Mon 01 Jan 2001 10:43:02 AM CST  
Mon 01 Jan 2001 08:43:02 AM PST
```

Note that the first alphabetical string is the zone name when not under daylight savings time, the digit is the number of hours displacement from GMT, and the second alphabetical string is the zone name when in daylight savings time. NOTE: This may not work on all operating systems.

2.81. timed_build

2.81.1. Summary

Parameters: **file**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[timed_build] FOO [/timed_build]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->timed_build(  
  {  
    file => VALUE,  
  },  
  BODY  
)
```

OR

```
$Tag->timed_build($file, $ATTRHASH, $BODY);
```

2.81.2. Description

Allows you to build CPU-intensive regions of ITL tags on a timed basis.

In the simplest case, surround a region of ITL with `[timed-build]` and `[/timed-build]`:

```
[timed-build]
    Built at [time]%c[/time].
[/timed-build]
```

If a `file` parameter is not passed, saves to the directory *timed* in catalog root, with the file name of the current page. If the `minutes` parameter is not passed specifying how often the page should be rebuilt, then it will not be rebuilt until the output file is removed.

Accepts the following parameters:

2.81.2.1. file

Name of the file to save the results in. Relative to catalog root. The directory must exist.

2.81.2.2. if

Allows you to only display the cached region when the `if` parameter is true. For example, you can do:

```
[timed-build if="[value timed]"]
ITL tags....
[/timed-build]
```

The cached region will only be displayed if the variable `timed` is set to a non-zero, non-blank value. Otherwise, the ITL tags will be re-interpreted every time.

2.81.2.3. minutes

The number of minutes after which the timed build should be repeated. If set to 0, it will be built once and then not rebuilt until the output file is removed.

2.81.2.4. period

Alternative way of expressing time. You can pass a string describing the rebuild time period:

```
[timed-build period="4 hours"]
ITL tags....
[/timed-build]
```

This is really the same as `minutes=240`. Useful for passing seconds:

```
[timed-build period="5 seconds"]
ITL tags....
[/timed-build]
```

The region will be rebuilt every 5 seconds. **PERFORMANCE TIP:** use minutes of `.08` instead; avoids having

to parse the period string.

If you have the StaticDir catalog.cfg parameter set to a writable path, you can build a cached static version of your catalog over time. Simply place a [timed-build] tag at the top of pages you wish to build statically. Assuming the catalog is not busy and write lock can be obtained, the StaticDBM database will be updated to mark the page as static and the next time a link is made for that page the static version will be presented.

2.82. tmp

2.82.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [tmp] FOO [/tmp]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->tmp(
    {
        name => VALUE,
    },
    BODY
)
```

OR

```
$Tag->tmp($name, $BODY);
```

2.82.2. Description

Sets a scratch variable to *value*, but at the end of the user session the Scratch key is deleted. This saves session write time in many cases.

This tag interpolates automatically. (Interpolation can be turned off with `interpolate=0`.)

IMPORTANT NOTE: the [tmp ...][/]tmp] tag is not appropriate for setting order profiles or mv_click actions. If you want to avoid that, use a profile stored via the catalog.cfg directive OrderProfile.

2.83. total_cost

2.83.1. Summary

Parameters: **name noformat**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->total_cost(
    {
        name => VALUE,
        noformat => VALUE,
    }
)
```

OR

```
$Tag->total_cost($name, $noformat);
```

Attribute aliases

```
cart ==> name
```

2.83.2. Description

Expands into the total cost of all the items in the current shopping cart, including sales tax (if any).

2.84. tree

2.84.1. Summary

Parameters: **table master subordinate start**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [tree] FOO [/tree]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->tree(
    {
        table => VALUE,
        master => VALUE,
        subordinate => VALUE,
        start => VALUE,
    },
    BODY
)
```

OR

```
$Tag->tree($table, $master, $subordinate, $start, $ATTRHASH, $BODY);
```

Attribute aliases

```
sub ==> subordinate
```

2.84.2. Description

Provides iterative list capability for binary trees. It produces hash-based rows use the same tags as [\[item-list\]](#); sets some additional hash key entries to describe the tree and provide display control.

Works on a data set with the structure:

parent	child
99	a
a	b
a	c
a	d
a	x
x	y
x	z
99	m
99	n
99	o
o	e
o	f
o	g

Sets several keys which assist in walking and displaying the tree.

2.84.2.1. mv_level

Level of the item. If it is in the first level, it is 0. Sublevels are infinite (except for performance).

2.84.2.2. mv_increment

Increment label for the item. Normally goes from 1...n, but can be changed to A...Z or a...z in outline mode.

2.84.2.3. mv_children

If in autodetect mode, set to the number of children this branch has. If a leaf, set to 0.

2.84.2.4. mv_spacing

A multiple of level times the spacing option. Useful for setting width of spacer images.

The above sample data would produce:

```

a      mv_level=0, mv_increment=1, mv_children=4
  b      mv_level=1, mv_increment=1, mv_children=0
  c      mv_level=1, mv_increment=2, mv_children=0
  d      mv_level=1, mv_increment=3, mv_children=0
  x      mv_level=1, mv_increment=4, mv_children=2
    y      mv_level=2, mv_increment=1, mv_children=0
    z      mv_level=2, mv_increment=2, mv_children=0
m      mv_level=0, mv_increment=1, mv_children=0
n      mv_level=0, mv_increment=2, mv_children=0
o      mv_level=0, mv_increment=3, mv_children=3
  e      mv_level=1, mv_increment=1, mv_children=0
  f      mv_level=1, mv_increment=2, mv_children=0
  g      mv_level=1, mv_increment=3, mv_children=0

```

from the tag call:

```

<table>
[tree  start=99
  parent=parent_fld
  child=child_fld
  autodetect=1
    spacing=4
  full=1]
<tr>
<td>
[if-item-param mv_level]
  [item-calc]
    return '&nbsp;' x [item-param mv_spacing];
  [/item-calc]
[/if-item-param]
[item-param child_fld]
</td>
<td>
  mv_level=[item-param mv_level],
  mv_increment=[item-param mv_increment],
  mv_children=[item-param mv_children]
</td>
</tr>
[/tree]
</table>

```

Accepts the following parameters:

2.84.2.5. table

Database table which contains the tree. Must be a valid Interchange table identifier.

2.84.2.6. parent

The column which is used to determine the parent of the item.

2.84.2.7. subordinate

The child column, which determines which items are sub-items of the current one. Used to re-query for items with its value in `parent`.

2.84.2.8. start_item

The first item to be followed, i.e. the `parent` value of all the top-level items.

2.84.2.9. autodetect

Specifies that the next level should be followed to detect the number of child items contained. Not recursive; only follows far enough to determine the children of the current item.

2.84.2.10. full

Specifies that all items should be followed. Essentially the same as specifying `memo` and passing the `explode` variable, but not dependent on them. Useful for building lists for inclusion in embedded Perl, among other things.

2.84.2.11. stop

An optional `stop` field which, when the value is true, can stop the following of the branch.

2.84.2.12. continue

An optional `continue` field which, when the value is true, can force the branch to be followed.

2.84.2.13. sort

The column which should be used for ordering the items — determines the order in which they will be displayed under the current parent.

2.84.2.14. outline

Sets outline mode, where `mv_increment` will be displayed with letter values or numeral values. If set to specifically 1, will produced outline increments like:

```

1
  A
  B
    1
    2
  C
    1
    2
      a
      b
        1

```

2

a
b

2

2.84.2.15. memo

Indicates that the collapse/explode/toggle features are to be used, and names a `Scratch` variable where the values should be stored.

2.84.2.16. collapse

The name of a variable in the user's session which will determine that the tree should be "collapsed". When collapsed, the child items will not be followed unless they are set to be followed with `toggle`. Zeros all toggles.

Requires `memo` to be set if values are to be retained.

2.84.2.17. toggle

The name of a variable in the user's session which will determine that the current item should be either followed or not followed. The first time the `toggle` variable corresponding to its primary key is passed, the item will be expanded. The next call will "collapse" the item.

Requires `memo` to be set if values are to be retained.

2.84.2.18. explode

The name of a variable in the user's session which will determine that the tree should be "exploded". When exploded, all child items are followed and the full tree can be displayed.

Requires `memo` to be set if values are to be retained.

2.84.2.19. pedantic

When set to a true value, and an endless tree is detected (i.e. the child branch contains a parent) then the error will be logged to the catalog error log and the tree call will return with an error.

If `pedantic` is not set (the default), the current leaf will be shown but never followed. This allows partial display of the tree.

2.84.2.20. log_error

When set to a true value, and an endless tree is detected (i.e. the child branch contains a parent) then the error will be logged to the catalog error log. No logging done by default.

2.84.2.21. show_error

When set to a true value, and an endless tree is detected (i.e. the child branch contains a parent) then the error will be returned in the page. Errors are NOT shown by default.

In addition to the above values, all valid options for a list tag are in force. For example, you can set a

"SELECTED" value on an option list with `option=1`, set the tag prefix with `prefix`, etc.

2.85. try

2.85.1. Summary

Parameters: **label**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

This is a container tag, i.e. `[try] FOO [/try]`. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP/perl tag calls:

```
$Tag->try(  
  {  
    label => VALUE,  
  },  
  BODY  
)
```

OR

```
$Tag->try($label, $ATTRHASH, $BODY);
```

2.85.2. Description

NO Description

2.86. update

2.86.1. Summary

Parameters: **function**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->update(  
    {  
        function => VALUE,  
    }  
)
```

OR

```
$Tag->update($function, $ATTRHASH);
```

2.86.2. Description

NO Description

2.87. userdb

2.87.1. Summary

Parameters: **function**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->userdb(  
    {  
        function => VALUE,  
    }  
)
```

OR

```
$Tag->userdb($function, $ATTRHASH);
```

Attribute aliases

```
name ==> nickname  
table ==> db
```

2.87.2. Description

Interchange provides a `[userdb ...]` tag to access the UserDB functions.

```
[userdb
  function=function_name
  username="username"*
  password="password"*
  verify="password"*
  oldpass="old password"*
  shipping="fields for shipping save"
  billing="fields for billing save"
  preferences="fields for preferences save"
  force_lower=1
  param1=value*
  param2=value*
  ...
]
```

* Optional

It is normally called in an `mv_click` or `mv_check` setting, as in:

```
[set Login]
mv_todo=return
mv_nextpage=welcome
[userdb function=login]
[/set]

<FORM ACTION="[process-target]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_click VALUE=Login>
Username <INPUT NAME=mv_username SIZE=10>
Password <INPUT NAME=mv_password SIZE=10>
</FORM>
```

There are several global parameters that apply to any use of the `userdb` functions. Most importantly, by default the database table is set to be *userdb*. If you must use another table name, then you should include a `database=table` parameter with any call to `userdb`. The global parameters (default in parens):

<code>database</code>	Sets user database table (<i>userdb</i>)
<code>show</code>	Show the return value of certain functions or the error message, if any (0)
<code>force_lower</code>	Force possibly upper-case database fields to lower case session variable names (0)
<code>billing</code>	Set the billing fields (see Accounts)
<code>shipping</code>	Set the shipping fields (see Address Book)
<code>preferences</code>	Set the preferences fields (see Preferences)
<code>bill_field</code>	Set field name for accounts (<i>accounts</i>)
<code>addr_field</code>	Set field name for address book (<i>address_book</i>)
<code>pref_field</code>	Set field name for preferences (<i>preferences</i>)
<code>cart_field</code>	Set field name for cart storage (<i>carts</i>)
<code>pass_field</code>	Set field name for password (<i>password</i>)
<code>time_field</code>	Set field for storing last login time (<i>time</i>)
<code>expire_field</code>	Set field for expiration date (<i>expire_date</i>)
<code>acl</code>	Set field for simple access control storage (<i>acl</i>)
<code>file_acl</code>	Set field for file access control storage (<i>file_acl</i>)
<code>db_acl</code>	Set field for database access control storage (<i>db_acl</i>)

2.88. value

2.88.1. Summary

Parameters: **name**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here.

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->value(
    {
        name => VALUE,
    }
)
```

OR

```
$Tag->value($name, $ATTRHASH);
```

2.88.2. Description

HTML examples:

```
<PARAM MV="value name">
<INPUT TYPE="text" NAME="name" VALUE="[value name]">
```

Expands into the current value of the customer/form input field named by field. If `flag` is present, single quotes will be escaped with a backslash; this allows you to contain the `[value . . .]` tag within single quotes. (It is somewhat better to use other quoting methods.) When the value is returned, any Interchange tags present in the value will be escaped. This prevents users from entering Interchange tags in form values, which would be a serious security risk.

If the `set` value is present, the form variable value will be set to it and the empty string returned. Use this to "uncheck" a checkbox or set other form variable values to defaults. **NOTE:** This is only available in new-style tags, for safety reasons.

2.89. value_extended

2.89.1. Summary

Parameters: **name**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter `interpolate=1` to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP/perl tag calls:

```
$Tag->value_extended(
    {
        name => VALUE,
    }
)
```

OR

```
$Tag->value_extended($name, $ATTRHASH);
```

2.89.2. Description

Named call example:

```
[value-extended
    name=formfield
    outfile=filename*
    ascii=1*
    yes="Yes"*
    no="No"*
    joiner="char|string"*
    test="isfile|length|defined"*
    index="N|N..N|*"
    file_contents=1*
    elements=1*]
```

Expands into the current value of the customer/form input field named by `field`. If there are multiple elements of that variable, it will return the value at `index`; by default all joined together with a space.

If the variable is a file variable coming from a multipart/form-data file upload, then the contents of that upload can be returned to the page or optionally written to the `outfile`.

2.89.2.1. name

The form variable NAME. If no other parameters are present, then the value of the variable will be returned. If there are multiple elements, then by default they will all be returned joined by a space. If `joiner` is present, then they will be joined by its value.

In the special case of a file upload, the value returned is the name of the file as passed for upload.

2.89.2.2. joiner

The character or string that will join the elements of the array. Will accept string literals such as `"\n"` or `"\r"`.

2.89.2.3. test

Three tests — `isfile` returns true if the variable is a file upload. `length` returns the length. `defined` returns whether the value has ever been set at all on a form.

2.89.2.4. index

The index of the element to return if not all are wanted. This is useful especially for pre-setting multiple search variables. If set to `*`, will return all (joined by `joiner`). If a range, such as `0 . . 2`, will return multiple elements.

2.89.2.5. file_contents

Returns the contents of a file upload if set to a non-blank, non-zero value. If the variable is not a file, returns nothing.

2.89.2.6. outfile

Names a file to write the contents of a file upload to. It will not accept an absolute file name; the name must be relative to the catalog directory. If you wish to write images or other files that would go to HTML space, you must use the HTTP server's `Alias` facilities or make a symbolic link.

2.89.2.7. ascii

To do an auto-ASCII translation before writing the `outfile`, set the `ascii` parameter to a non-blank, non-zero value. Default is no translation.

2.89.2.8. yes

The value that will be returned if a test is true or a file is written successfully. Defaults to 1 for tests and the empty string for uploads.

2.89.2.9. no

The value that will be returned if a test is false or a file write fails. Defaults to the empty string.

3. User-defined Tags

To define a tag that is catalog-specific, place *UserTag* directives in your `catalog.cfg` file. For server-wide tags, define them in `interchange.cfg`. Catalog-specific tags take precedence if both are defined — in fact, you can override the base Interchange tag set with them. The directive takes the form:

```
UserTag tagname property value
```

where `tagname` is the name of the tag, `property` is the attribute (described below), and `value` is the value of the property for that `tagname`.

The user tags can either be based on Perl subroutines or just be aliases for existing tags. Some quick examples are below.

An alias:

```
UserTag product_name Alias      data products title
```

This will change `[product_name 99-102]` into `[data products title 99-102]`, which will output the `title` database field for product code 99-102. Don't use this with `[item-data ...]` and `[item-field ...]`, as they are parsed separately. You can do `[product-name [item-code]]`, though.

A simple subroutine:

```
UserTag company_name Routine    sub { "Your company name" }
```

When you place a `[company-name]` tag in an Interchange page, the text `Your company name` will be substituted.

A subroutine with a passed text as an argument:

```
UserTag caps      Routine      sub { return "\U@" }
UserTag caps      HasEndTag
```

The tag `[caps]`This text should be all upper case[/caps] will become `THIS TEXT SHOULD BE ALL UPPER CASE`.

Here is a useful one you might wish to use:

```
UserTag quick_table HasEndTag
UserTag quick_table Interpolate
UserTag quick_table Order    border
UserTag quick_table Routine  <<EOF
sub {
    my ($border,$input) = @_;
    $border = " BORDER=$border" if $border;
    my $out = "<TABLE ALIGN=LEFT$border>";
    my @rows = split /\n+/, $input;
    my ($left, $right);
    for(@rows) {
        $out .= '<TR><TD ALIGN=RIGHT VALIGN=TOP>';
        ($left, $right) = split /\s*:\s*/ , $_, 2;
    }
}
```

```

    $out .= '<B>' unless $left =~ /</;
    $out .= $left;
    $out .= '</B>' unless $left =~ /</;
    $out .= '</TD><TD VALIGN=TOP>';
    $out .= $right;
    $out .= '</TD></TR>';
    $out .= "\n";
  }
  $out .= '</TABLE>';
}
EOF

```

Called with:

```

[quick-table border=2]
Name: [value name]
City: [value city][if value state], [value state][if] [value country]
[/quick_table]

```

The properties for UserTag are are:

3.1. Alias

An alias for an existing (or other user-defined) tag. It takes the form:

```
UserTag tagname Alias    tag to insert
```

An Alias is the only property that does not require a *Routine* to process the tag.

3.2. attrAlias

An alias for an existing attribute for defined tag. It takes the form:

```
UserTag tagname attrAlias  alias attr
```

As an example, the standard Interchange `value` tag takes a named attribute of `name` for the variable name, meaning that `[value name=var]` will display the value of form field `var`. If you put this line in `catalog.cfg`:

```
UserTag value attrAlias  identifier name
```

then `[value identifier=var]` will be an equivalent tag.

3.3. CanNest

Notifies Interchange that this tag must be checked for nesting. Only applies to tags that have *HasEndTag* defined, of course. NOTE: Your routine must handle the subtleties of nesting, so don't use this unless you are quite conversant with parsing routines. See the routines `tag_loop_list` and `tag_if` in `lib/Vend/Interpolate.pm` for an example of a nesting tag.

```
UserTag tagname CanNest
```

3.4. HasEndTag

Defines an ending [/tag] to encapsulate your text — the text in between the beginning [tagname] and ending [/tagname] will be the last argument sent to the defined subroutine.

```
UserTag tagname HasEndTag
```

3.5. Implicit

This defines a tag as implicit, meaning it can just be an attribute instead of an attribute=value pair. It must be a recognized attribute in the tag definition, or there will be big problems. Use this with caution!

```
UserTag tagname Implicit attribute value
```

If you want to set a standard include file to a fixed value by default, but don't want to have to specify [include file="/long/path/to/file"] every time, you can just put:

```
UserTag include Implicit file file=/long/path/to/file
```

and [include file] will be the equivalent. You can still specify another value with C[include file="/another/path/to/file"]

3.6. InsertHTML

This attribute makes HTML tag output be inserted into the containing tag, in effect adding an attribute=value pair (or pairs).

```
UserTag tagname InsertHTML    htmltag  mvtag|mvtag2|mvtagN
```

In Interchange's standard tags, among others, the <OPTION ...> tag has the [selected ..] and [checked ...] tags included with them, so that you can do:

```
<INPUT TYPE=checkbox
      MV="checked mvshipmode upsg" NAME=mv_shipmode> UPS Ground shipping
```

to expand to this:

```
<INPUT TYPE=checkbox CHECKED NAME=mv_shipmode> UPS Ground shipping
```

Providing, of course, that mv_shipmode is equal to upsg. If you want to turn off this behavior on a per-tag basis, add the attribute mv.noinstert=1 to the tag on your page.

3.7. InsideHTML

To make a container tag be placed **after** the containing HTML tag, use the InsideHTML setting.

```
UserTag tagname InsideHTML    htmltag  mvtag|mvtag2|mvtagN
```

In Interchange's standard tags, the only InsideHTML tag is the <SELECT> tag when used with *loop*, which

causes this:

```
<SELECT MV="loop upsg upsb upsr" NAME=mv_shipmode>
<OPTION VALUE="[loop-code]"> [shipping-desc [loop-code]]
</SELECT>
```

to expand to this:

```
<SELECT NAME=mv_shipmode>
[loop upsg upsb upsr]
<OPTION VALUE="[loop-code]"> [shipping-desc [loop-code]]
[/loop]
</SELECT>
```

Without the `InsideHTML` setting, the `[loop ...]` would have been **outside** of the select — not what you want. If you want to turn off this behavior on a per-tag basis, add the attribute `mv.noinside=1` to the tag on your page.

3.8. Interpolate

The behavior for this attribute depends on whether the tag is a container (i.e. `HasEndTag` is defined). If it is not a container, the `Interpolate` attribute causes the **the resulting HTML** from the `UserTag` to be re-parsed for more Interchange tags. If it is a container, `Interpolate` causes the contents of the tag to be parsed **before** the tag routine is run.

```
UserTag tagname Interpolate
```

3.9. InvalidateCache

If this is defined, the presence of the tag on a page will prevent search cache, page cache, and static builds from operating on the page.

```
UserTag tagname InvalidateCache
```

It does not override `[tag flag build][tag]`, though.

3.10. Order

The optional arguments that can be sent to the tag. This defines not only the order in which they will be passed to *Routine*, but the name of the tags. If encapsulated text is appropriate (*HasEndTag* is set), it will be the last argument.

```
UserTag tagname Order param1 param2
```

3.11. PosRoutine

Identical to the *Routine* argument — a subroutine that will be called when the new syntax is not used for the call, i.e. `[usertag argument]` instead of `[usertag ARG=argument]`. If not defined, *Routine* is used, and Interchange will usually do the right thing.

3.12. ReplaceAttr

Works in concert with InsertHTML, defining a **single** attribute which will be replaced in the insertion operation..

```
UserTag tagname ReplaceAttr htmltag attr
```

An example is the standard HTML tag. If you want to use the Interchange tag [area pagename] inside of it, then you would normally want to replace the HREF attribute. So the equivalent to the following is defined within Interchange:

```
UserTag area ReplaceAttr a href
```

Causing this

```
<A MV="area pagename" HREF="a_test_page.html">
```

to become

```
<A HREF="http://yourserver/cgi/simple/pagename?X8sl2lly;;44">
```

when interpreted.

3.13. ReplaceHTML

For HTML-style tag use only. Causes the tag containing the Interchange tag to be stripped and the result of the tag to be inserted, for certain tags. For example:

```
UserTag company_name Routine sub { my $l = shift; return "$l: XYZ Company" }
UserTag company_name HasEndTag
UserTag company_name ReplaceHTML b company_name
```


 is the HTML tag, and "company_name" is the Interchange tag. At that point, the usage:

```
<B MV="company-name"> Company </B> ----> Company: XYZ Company
```

Tags not in the list will not be stripped:

```
<I MV="company-name"> Company </I> ----> <I>Company: XYZ Company</I>
```

3.14. Routine

An inline subroutine that will be used to process the arguments of the tag. It must not be named, and will be allowed to access unsafe elements only if the interchange.cfg parameter *AllowGlobal* is set for the catalog.

```
UserTag tagname Routine sub { "your perl code here!" }
```

The routine may use a "here" document for readability:

```
UserTag tagname Routine <<EOF
```

Interchange Tags Reference

```
sub {  
    my ($param1, $param2, $text) = @_;  
    return "Parameter 1 is $param1, Parameter 2 is $param2";  
}  
EOF
```

The usual *here documents* caveats apply.

Parameters defined with the *Order* property will be sent to the routine first, followed by any encapsulated text (*HasEndTag* is set).

Note that the UserTag facility, combined with AllowGlobal, allows the user to define tags just as powerful as the standard Interchange tags. This is not recommended for the novice, though — keep it simple. 8—)

4. Standard Uertags

The distribution includes a number of prebuilt uertags in the `interchange_root/uertag` directory. Some of these are used by the foundation catalog or its administrative interface.

4.1. bar_button

4.2. delete_cart

4.3. fedex_query

4.4. loc

4.5. reconfig_wait

4.6. title_bar

4.7. xml_generator

4.8. button

4.9. email

4.10. formel

4.11. rand

4.12. save_cart

4.13. ups_query

4.14. convert_date

4.15. email_raw

4.16. get_url

4.17. reconfig

4.18. summary

4.19. usertrack

4.20. db_date

4.21. fcounter

4.22. load_cart

4.23. reconfig_time

4.24. table_organize

4.25. var

A. Tag Entry Format

4.26. dummy

Alias: **pedagogy**, **dummy_alias**

Note: [**dummy** ...], [**pedagogy** ...] and [**dummy--alias** ...] are equivalent.

A short description of the tag goes here. This example (**dummy**) is not an actual tag.

A..1. Summary

```
[dummy first second other named attributes][/dummy]  
[dummy first=first_args second=second_args other named attributes][/dummy]
```

Positional parameters The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	Description	Default
first	The first positional parameter <i>Special arguments</i> <ul style="list-style-type: none">◆ 'special_value' -- any special arguments to the parameter that cause the tag to behave differently are listed here and described in detail in the Description section below.◆ 'pig_latin' -- For this dummy tag, let's suppose that an argument of 'pig_latin' rewrites the body text in pig latin.	default value if the parameter is not given
second	Another example parameter	<i>none</i>
<i>alias1</i>	alias for first -- some parameters have aliases -- [dummy alias1="X"] is equivalent to [dummy first="X"]	Same default as first , of course

Attributes	Default
more	<i>none</i>
still-more	<i>none</i> (requires more)
other	<i>none</i>
interpolate (body)	<i>Yes</i>
reparse	<i>No</i>

<i>Other_Charactreristics</i>	
-------------------------------	--

Invalidates cache	<i>No</i>
Macro	<i>No</i>
Has end tag	[/dummy]

Tag expansion example:

```
[dummy first=pig_latin second="Capitalize"]\
Body text acted on by the tag goes here.[/dummy]
---
OdyBay ExtTaY acted on YBay EThay AgTay OesGay \
EreHay.
```

Reading the tag expansion example:

format: The tag is listed first. A blank line separates it from the expanded return value.

long lines: When this document must break a line from an example because it is too wide for the page, a trailing backslash indicates the continuation. Note that such a trailing backslash is **not** part of the actual tag syntax or expansion.

ASP-like Perl call

```
$Tag->area( { href => URI_base,
             arg  => CGI_GET_arguments, } );
```

or similarly,

```
$Tag->area($href, $arg, $attribute_hash_reference);
```

A..2. Description

More detailed tag description

A..2.1. first

Section describing the first parameter

A..2.1.1. special_value

Description of treatment of the special argument. For example, giving the [page](#) tag an **href** of 'scan' causes it to link to a search specification rather than a page.

A..2.2. more

`more` is a named attribute that applies to this tag.

A..2.3. still-more

`still-more` is a named attribute that applies to this tag only when the `more` attribute has been given. It would be an error to use the `still-more` attribute without specifying a value for `more`

A..2.4. other

'other' is another named attribute that applies to this tag.

B. Template Parsing Order

Under normal circumstances, the template page containing tags and HTML is parsed in the following order:

1. Any content in MV_AUTOLOAD is prepended to the template page.
2. Any [\[pragma\]](#) tags anywhere in the text are processed, and the specified pragmas are set.
 - ◆ Since [\[pragma\]](#) tags are preprocessed before any other content, [reparse](#) will not catch them, nor will they work if included in variables. Also, the specified pragma will apply to the entire template (not just the section after the tag).
 - ◆ If you want to apply a pragma with a variable or only to part of a document, you must use [\[tag pragma="..."\]](#) instead.
3. Variables (macros) are processed in the following order:
 1. @@VARNAME@@ global variables
 2. @_VARNAME_@ local or global variables
 3. __VARNAME__ local variables
4. Interchange comments are stripped.
5. False–endtag macros are expanded (e.g., [\[/page\]](#) and [\[/order\]](#)).
6. '<!--[tagname]-->' escapes are converted to [\[tagname\]](#)
 - ◆ This can be a convenience for your HTML editor if it has trouble with tags using the standard [\[tagname\]](#) syntax.
 - ◆ However, if you want to HTML–comment out an interchange tag in content that will be fed raw to a browser, you must include whitespace, like this, '<!-- [tagname] -->'.
7. The main tag parser is called.
 - ◆ Some tags parse recursively (depending on [reparse](#) and [interpolate](#) settings, of course).
 - ◆ Some tags (e.g., [\[loop\]](#)) process prefix–tags in their contained body text. The prefix–tags are not handled recursively.
8. Image paths substitution on the HTML output occurs.

Nonstandard parsing within the admin interface

Parsing of content via the specialized [regenerate](#) usertag included with the administrative interface does not obey the above order. The MV_AUTOLOAD and '<!--[tagname]-->' escapes are skipped. There are some other more subtle differences as well; in the very unlikely event that you need to check this in the source code, compare the 'interpolate_html' and 'cache_html' routines in Interpolate.pm.
