# Interchange Tags Reference

# Table of Contents

# Table of Contents

# Table of Contents

# 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 that perform your own functions and can be just as powerful as the built−in tags.

This document covers Interchange versions 4.7−4.8.

## 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 </A>. 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 completely interchangeable, the Interchange parser does not replace ']−−>' with ']' unless it also sees '<!−−[' at least once somewhere on the page. (This is a small parsing speed optimization.)

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 name=value pair just as most HTML tags use:

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

The positional-style tag that accomplishes the same thing looks like this:

```
[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, tag parameters specified in the positional fashion work the same as named parameters. However, there are a few situations where you need to use named parameters:

1. If you want to specify a parameter that comes positionally after a parameter that you want to omit, e.g. omit the first parameter but specify the second. The parser would have no way of knowing which is which, so you just specify the second by name. This is rare, though, because the first positional parameters are usually required for a given tag anyway.
2. When there is some ambiguity as to which parameter is which, usually due to whitespace.
3. 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.

### 1.1.6. Attribute Arrays and Hashes

Some tags allow you to pass an array or hash as the value of an attribute. For an ordinary tag, the syntax is as follows:

```
attribute.n=value

attribute.hashkey=value
```

where *n* is an integer array index. Note that you cannot use both an array and a hash with the same attribute −− if you use **attribute.*n***, you cannot also use **attribute.key** for the same attribute.

Here is an example of an attribute array:

```
search.0="se=hammer
          fi=products
          sf=description"
search.1="se=plutonium
          fi=products
          sf=comment"
```

The [page] tag, for example, treats a search specification array as a joined search, automatically adding the other relevant search fields, including the 'co=yes' to indicate a combined search (joined searches are described in the Interchange database documentation).

Note that it is up to the tag to handle an array or hash value properly. See the documentation for the specific tag before passing it an attribute array or hash value.

#### 1.1.6.1. Perl calls

Before passing attributes to a tag, the Interchange parser would convert the above example to an anonymous array reference. It would use the resulting arrayref as the value for the 'search' attribute in this example.

If you were passing the above example directly to a tag routine within a [perl] block or a usertag, you must actually pass an anonymous array as the value of the attribute as follows:

```
my $arrayref = [ "se=hammer/fi=products/sf=description",
                 "se=plutonium/fi=products/sf=description", ];

$Tag->routine( { search => $arrayref, } );
```

Similarly to use a hash reference for the 'entry' attribute:

```
my $hashref = { name   => "required",
                date   => 'default="%B %d, %Y"', };

$Tag->routine( { entry => $hashref } );
```

## 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]
[PREFIX-calc]
[PREFIX-change]
[PREFIX-code]
[PREFIX-data]
[PREFIX-description] (Note safe-data and ed( ) escape)
[PREFIX-discount]
[PREFIX-discount_subtotal]
[PREFIX-exec]
[PREFIX-field] (Optimization note-- one query per field if you use this; we optimize around this if only one products table)
[PREFIX-filter] (like filter tag but doesn't interpolate)
[PREFIX-increment]
[PREFIX-last]
[PREFIX-line] (tab-delimited list of parameters returned, can do tail-slice)
[PREFIX-match]
[PREFIX-modifier]
[PREFIX-next]
[PREFIX-param]
[PREFIX-options]
[PREFIX-price]
[PREFIX-quantity]
[PREFIX-sub]
[PREFIX-subtotal]
[if-PREFIX-data]
[if-PREFIX-field]
[if-PREFIX-param]
[if-PREFIX-modifier] (hash list only)
[if-PREFIX-pos]
[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:

| *Tag* | *Prefix* | *Examples* |
|---|---|---|
| [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. Parse Order

Subtags are parsed during evaluation of the enclosing loop, *before* any regular tags within the loop

### 1.2.2. [PREFIX−accessories arglist]

Except for the usual differences that apply to all subtags (such as parsing order), these are more or less equivalent for an array−type list:

```
[accessories code=current_item_code arg=arglist]
[item−accessories arglist]
```

See the accessories tag for more detail. Note that you must use the comma−delimited list to set attributes −− you cannot set named attributes with the usual 'attribute=value' syntax.

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.

### 1.2.3. [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]
```

### 1.2.4. [PREFIX−calc] 2 + [item−field price] [/PREFIX−calc]

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

### 1.2.5. [PREFIX−change][condition] ... [/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]
                 
        [/else]
        [/item-change cat]
   </TD>
   <TD>
        [item-change sub]

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

                [item-field subcategory]
        [else]
                 
        [/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.)

## 1.2.6. [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).

## 1.2.7. [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`.

## 1.2.8. [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.

## 1.2.9. [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.

## 1.2.10. [PREFIX−discount_subtotal]

Inserts the discounted subtotal of the ordered items.

## 1.2.11. [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`.

## 1.2.12. [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.

## 1.2.13. [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.

## 1.2.14. [PREFIX−modifier attribute]

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

## 1.2.15. [PREFIX−next] CONDITION [/PREFIX−next]

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

## 1.2.16. [PREFIX−param name]

## 1.2.17. [PREFIX−pos 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.

[PREFIX−pos N] outputs the *N*th field as returned; [PREFIX−param] returns it by name.

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−pos 0]; and [prefix−pos 1] is the same as [prefix−param bar].

## 1.2.18. [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).

## 1.2.19. [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].

## 1.2.20. [PREFIX−subtotal]

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

## 1.2.21. [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!

### 1.2.22. [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]`.

### 1.2.23. [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.

### 1.2.24. [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

A Swiss−army−knife widget builder, this provides 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), forms or hyperlinks, or can simply return a value.

Or more −− see also Looping tags and Sub−tags.

### 2.1.1. Summary

```
[accessories code arg]
[accessories code=os28044 arg="size, radio, ... " other_named_attributes] deprecated
[accessories code=os28044 attribute=size type=radio ... other_named_attributes]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| code | Value of the master key in the product (or specified other) table | *none* |
| arg | Positionally interpreted comma−delimited list of values for the following attributes:<br><br>"attribute, type, column, table, name, outboard, passed" | *none* |

| *Attributes* | *Default* |
|---|---|
| attribute | *none* |
| type<br><br>One of select, value, text, textarea, hidden, password, combo, move_combo, reverse_combo, show, options, labels, checkbox, radio, links | select |
| column | *attribute* |
| table | products |
| name | mv_order_*attribute* |
| outboard | *none* |
| passed | *none* |
| key (alias for code) | *none* |
| row (alias for code) | *none* |
| base (alias for `table`) | products |
| database (alias for `table`) | products |
| db (alias for `table`) | products |
| col (alias for `column` | *attribute* |
| field (alias for `column` | *attribute* |
| delimiter | comma (',') |

| | |
|---|---|
| prepend | *none* |
| append | *none* |
| extra | *none* |
| js | *none* |
| rows | *varies with* `type`*; often* 4 |
| cols | *varies with* `type`*; often* 40 |
| width | *none* |
| default | *none* |
| price | *none* |
| price_data | *none* |
| contains (`type`=radio or check) | *none* |
| joiner (`type`=links) | *none* |
| href (`type`=links) | *none* |
| template (`type`=links) | *none* |
| form (`type`=links) | `mv_action=return` |
| empty (`type`=links) | *none* |
| secure (`type`=links) | *none* |
| new | *none* |
| interpolate (reparse) | *No* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Container tag | *No* |
| Has Subtags | *No* |

**Tag expansion example:**

```
    [accessories os28044 size]
----------------------------------------------------------------
    <SELECT NAME="mv_order_size"><OPTION VALUE="10oz">10oz\
    <OPTION VALUE="15oz">15oz<OPTION VALUE="20oz">20oz</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

Looping tags and Sub−tags.

## 2.1.2. Description

This is the swiss−army−knife widget builder for providing access to Interchange's product option attributes (e.g., to choose or access product options such as a shirt's size or color).

Interchange allows you to choose item attribute values for each ordered item −− you can attach a size, color, or other modifier to a line item in the shopping cart. You can also resubmit previous attribute values via hidden fields 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 modifier names 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 will automatically generate the select boxes when the [`accessories` code=os28044 attribute=size] or `[item−accessories size]` tags are called. They have the syntax:

```
[item−accessories attribute, type, column, table, name, outboard, passed]

[accessories code=sku
             attribute=modifier
             type=select
             column=db_table_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]" ]
```

---

***Notes:***

1. The '`attribute`' attribute is required.
2. See the `type` attribute for a list of types.
3. The trailing '*' in value2 will mark it as the default ('SELECTED') value in the select widget (see below).

---

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

```
name_a=Label Text1, default_name=Default Label Text*, name_b, etc.
```

The label text is optional -- if none is given, the **name** will be used (as in 'name_b' above).

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` session variables, you can use this to allow a customer to enter an item attribute during an 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` color] on the order report, order receipt, or any other page containing an [`item-list`].

#### 2.1.2.1. Emulating with a loop

You can also build widgets directly, without using the `accessories` tag. You may have to do so if you need more control of the content than the tag offers. 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 example 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 output of the above would be similar to the output of [item-accessories size, select] if the product database field `size` contained the value `S, M, L, XL`. The difference is that the options in the loop emulation show the adjusted price in addition to the size within each option value.

#### 2.1.2.2. Hash Lists -- Technical Note

As a technical note, some of the features of this tag work differently depending on whether it was called with an '$item' hash reference, for example, as [`item-accessories`] within an [`item-list`].

In this context, the tag will have access to ancillary data from the item (including, perhaps, a user's chosen item attribute value). For example, if building a TEXTAREA widget within an [`item-list`], the widget will show the chosen item attribute value. On the other hand, within an array list such as a [`search-list`] in a [`search-region`], the widget would be empty.

If you really know what you're doing, you can pass it the item hash reference within a `perl` tag like this:

```
$Tag->accessories( $code,
                   undef,              # 'arg' parameter value
                   $named_attribute_hashref,
                   $item_hashref );
```

See also `Looping tags and Sub-tags` for information about hash- and array-context in looping tags.

#### 2.1.2.3. code

This is the master key of the specified table (commonly sku in a product table). If no table is specified, the tag uses the products table by default.

You should not specify a `code` when looping on [item_accessories] because it internally sets '`code`' to the key for the current item in the loop.

#### 2.1.2.4. arg

**Deprecated after Interchange 4.6**

This allows you to pass values for some of the more commonly used attributes in the manner of the [`PREFIX-accessories`] tag, as a comma-delimited positional list:

```
arg="attribute, type, column, table, name, outboard, passed"
```

Whitespace within the list is optional.

If you leave out one or more of the above attributes, be sure to keep the comma(s) if you are setting anything after it in the list:

```
arg="attribute, type, , table"
```

The above examples show the attribute names for clarity; you would actually use the values. Hence, the

previous example might actually be something like the following:

```
arg="color, radio, , products"
```

Although you must use such a comma−delimited list to pass attributes to the [PREFIX−accessories] tag, please use named attributes instead for the [accessories] tag. The 'arg' attribute is deprecated.

For detail about a specific attribute, please see its subheading below.

#### 2.1.2.5. attribute

Despite the name, this has nothing to do with tag attributes. You can set attributes for *items* in a database table (typically the products table) with the UseModifier configuration directive. Typical are size or color.

This selects the item attribute the tag will work with.

#### 2.1.2.6. type

This determines the action to be taken. One of:

| ***Action*** | ***Description*** |
|---|---|
| select | Builds a dropdown <SELECT> menu for the item attribute, with the default item attribute value SELECTED. The accessories tag builds a select widget by default if type is not set. |
| display | Shows the label text for *only the selected option* if called in Hash List context (e.g., within an [item−list]). Ignored otherwise (i.e., the tag will build the default <SELECT> menu). |
| show | Returns the list of possible attributes for the item (without labels or any HTML widget). For example, if sku os28044 is available in several sizes:<br><br>[accessories os28044 size,show]<br>−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−<br>Sm=10oz, Med=15oz*, Lg=20oz |
| options | This shows the attribute options as a newline delimimited list:<br><br>[accessories os28044 size,options]<br>−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−<br>Sm<br>Med<br>Lg |
| labels | This shows the attribute option labels:<br><br>[accessories os28044 size,options]<br>−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−<br>10oz<br>15oz*<br>20oz |
| 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 break | Builds a radio box group for the item, with '<br>' separating the radio button/label pairs from one another. |
| 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.<br><br>You can also set FONT SIZE like this:<br><br>`type="radio left n fontsizem"`<br><br>where −9 <= *m* <= 9 |
| 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.<br><br>You can also set FONT SIZE like this:<br><br>`type="radio right n fontsizem"`<br><br>where −9 <= *m* <= 9 |
| 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 checkbox/label pairs from one another. |
| check break | Builds a checkbox group for the item, with '<br>' separating the checkbox/label pairs from one another. |
| 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.<br><br>You can also set FONT SIZE like this:<br><br>`type="check left n fontsizem"`<br><br>where −9 <= *m* <= 9 |
| 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.<br><br>You can also set FONT SIZE like this:<br><br>`type="check right n fontsizem"`<br><br>where −9 <= *m* <= 9 |
| textarea_XX_YY | A textarea with XX columns and YY rows. The textarea will contain the selected item attribute value if used in `Hash List` context (e.g., within an [`item-list`]).<br><br>If you simply use '`type`=textarea', the size will default to 4 rows by 40 columns, unless you have set the `rows` or `cols` tag attributes. |
| text_YY | A text box with YY width in characters. The HTML tag's VALUE will be set to the selected item attribute value if used in `Hash List` context (e.g., within an [`item-list`]). |

| | |
|---|---|
| | If you simply use 'type=text', the width will default to 60, unless you have set the cols tag attribute. |
| 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. |
| value | Returns the selected value if called in Hash List context (e.g., within an [item−list]), or nothing otherwise. |
| hidden | Creates a hidden form field. The hidden field's VALUE will be set to the selected item attribute value if used in Hash List context (e.g., within an [item−list]). |
| password_YY | A password box with YY width in characters. The HTML tag's VALUE will be set to the selected item attribute value if used in Hash List context (e.g., within an [item−list]).<br><br>If you simply use 'type=password', the width will default to 12, unless you have set the cols tag attribute. |

The default is 'select', which builds an HTML select form entry for the attribute.

Some types build widgets that use the ROWS=*m*, COLS=*n*, or certain other HTML attributes. For these, you can define widget rows and columns within the string that sets the type; for example, type="textarea_6_33_wrap=virtual" specifies a TEXTAREA widget with ROWS=6, COLS=33, and WRAP=virtual. You should resort to this only when you cannot use the named parameters, for example within an [item−accessories] tag. Otherwise, use the rows=*m* and cols=*n* tag attributes instead.

The result of setting conflicting values in the type string and the rows or cols attributes is undefined.

The following list shows syntax for type strings, where *rows* is the number of rows and *cols* is the number of columns.

- **text**
  - textarea *(default is 4 rows, 40 columns, like 'textarea_4_40')*
  - textarea_*rows_cols*
  - text_*cols*
  - textarea rows=*rows* cols=*cols* wrap=*WRAP value*
- **password**
  - password *(default is 12 columns, like 'password_12')*
  - password_*cols*
- **combo** (similarly for **reverse_combo** and **move_combo**)
  - combo *(default is 1 row, 16 columns, like 'combo_1_16')*

In any of the option building types, you can append the string ranges and a special option processing will be done −− any option matching the pattern [A−Za−z0−0]..[A−Za−z0−0] will be expanded into a comma separated range between the bounds. The same behavior is accomplished by passing the accessories tag option ranges. For example:

```
[accessories name=foo type=select ranges=1 "A..C,1..5,10,20"]
  and
[accessories name=foo type="select ranges" passed="A..C,1..5,10,20"]

  will both output:

<select NAME="foo">
<option VALUE="A">A
<option VALUE="B">B
<option VALUE="C">C
<option VALUE="1">1
<option VALUE="2">2
<option VALUE="3">3
<option VALUE="4">4
<option VALUE="5">5
<option VALUE="10">10
<option VALUE="15">15
<option VALUE="20">20
</select>
```

The above applies to any of the option building types −− `check`, `combo`, `combo_move`, `labels`, `multiple`, `options`, `radio`, `reverse_combo`, and `select`. It will refuse to produce more than 5000 options −− that limit can be changed with `Limit option_list N` in `catalog.cfg`, where N is an integer greater than 0.

#### 2.1.2.7. column

The column of the table corresponding to the attribute will traditionally have the same name as the attribute, though it need not.

This specifies the table column that contains an item's attribute values. The tag will find item attribute names and values in a comma−delimited list of name=value pairs stored in this field of an item's table entry. If unspecified, the column name will default to the name given for the '`attribute`' attribute.

For example, if an item in the products table has a '`size`' attribute, and each item's comma−delimited list of available sizes is stored in the '`how_big`' column, then you would need to specify "`column=how_big`" because the tag's default column choice (`size`) would be missing or used for some other purpose.

#### 2.1.2.8. table

This is the database table containing the item's attribute values. It defaults to the first products file where the item code is found.

If you have configured your database so that the attributes are kept in a different table from other item data, '`code`' should be set to the master key in this table. See '`outboard`') if you are using [item−accessories ...] and cannot specify `code`=key.

#### 2.1.2.9. name

This sets the name of the form variable to use if appropriate for the widget 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. In other words,

[value *name*] will contain the previous setting, which the widget will use as its default setting. See also the default attribute.

#### 2.1.2.10. outboard

If calling the item−accessories tag, and you wish to select from an outboard database table whose master key is different from the item code, you can pass the key the tag should use to find the accessory data.

#### 2.1.2.11. passed

You can use this to pass your own values to the widget the tag will build. If you have set passed to a list of widget options, then the tag will simply build a widget of the specified type with your values instead of fetching an attribute value list from the database.

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="" SELECTED>−−select−−\
<OPTION VALUE="blue">Blue\
<OPTION VALUE="green">Sea Green</SELECT>
```

Note: trailing backslashes ('\') in the above example indicate line continuation and are not part of the tag output.

#### 2.1.2.12. delimiter

The list of attribute values will be a delimited string. This allows you to specify an alternative delimiter if the list is not comma−delimited (the default).

#### 2.1.2.13. prepend

You can set a string to prepend to the returned output of the tag. Note that this is *not* a list to prepend to the fetched attribute value list, which is treated within the tag.

For example,

```
   [accessories code=os28044
                type=select
           attribute=size
              append="Append Me<br>"
             prepend="Prepend Me"]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
   Prepend Me<SELECT NAME="mv_order_size">\
   <OPTION VALUE="10oz">10oz\
   <OPTION VALUE="15oz">15oz\
   <OPTION VALUE="20oz">20oz</SELECT>Append Me<br>
```

#### 2.1.2.14. append

You can set a string to append to the returned output of the tag. Note that this is *not* a list to append to the fetched `attribute` value list, which is treated within the tag.

#### 2.1.2.15. extra

Setting the '`extra`' attribute appends its value as the last attribute of the HTML output tag. The following example illustrates the `append`, `extra` and `js` options:

```
   [accessories code=os28044
                type=select
           attribute=size
              append="Append Me<br>"
               extra="Last=Extra"
                  js="javascript_here"]
-----------------------------------------------------
   <SELECT NAME="mv_order_size" javascript_here Last=Extra>\
   <OPTION VALUE="10oz">10oz\
   <OPTION VALUE="15oz">15oz\
   <OPTION VALUE="20oz">20oz</SELECT>Append Me<br>
```

#### 2.1.2.16. js

This allows you to place javascript within the start tag of the HTML output. See the example given above for `extra`.

`js` has no default, except when '`type`=move_combo', where the default is:

```
  onChange="addItem(this.form.X name,this.form.name)"
```

#### 2.1.2.17. rows

The tag will pass the number you choose through to the HTML 'ROWS=*n*' attribute in HTML widgets that accept it.

For some types, you can also define widget rows and columns within the string that sets the `type`; for example, `type`="textarea_6_33_wrap=virtual" specifies a TEXTAREA widget with ROWS=6, COLS=33, and WRAP=virtual. You should resort to this only when you cannot use the named parameters, for example within an [`item-accessories`] tag.

The result of setting conflicting values in the `type` string and the `rows`=*n* attribute is undefined.

#### 2.1.2.18. cols

The tag will pass the number you choose through to the HTML 'COLS=*n*' attribute in HTML widgets that accept it.

See also '`rows`' above.

#### 2.1.2.19. width

This is a quasi−alias for '`cols`' that only works with the '`text`' and '<password>' types. Use '`cols`' instead.

#### 2.1.2.20. default

Sets the default attribute option in the widget returned by the tag. This will override a default indicated with a trailing '*' in the database or '`passed`' string. This will also override the default of a user's previous selection when the tag would otherwise have preserved it.

For example the following selects blue by default rather than green as it would otherwise have done,

```
  [accessories type=select
               name=color
             passed="blue=blue, green=Sea Green*"
            default="blue"]
------------------------
  <SELECT NAME="color"><OPTION VALUE="blue" SELECTED>blue\
  <OPTION VALUE="green">Sea Green</SELECT>
------------------------
```

Obscure technical note: the tag ignores the '`default`' attribute if it has an item hash reference −− see `Hash Lists` above.

#### 2.1.2.21. price

When combined with the `price_data` tag attribute, this allows you to force prices for item attributes. You probably do not want to use this; just let the tag pick up prices from your database table(s) when appropriate.

If you are passing attribute values, you can use this to control the displayed price in the widget.

```
  [accessories type=check
               name=color
              price=1
         price_data="blue=20, green=50"
             passed="blue=Blue, green=Sea Green*" ]
-------------------------------------------------
  <INPUT TYPE="checkbox" NAME="color" VALUE="blue" > Blue ($20.00)
  <INPUT TYPE="checkbox" NAME="color" VALUE="green" CHECKED> Sea Green ($50.00)
```

#### 2.1.2.22. contains

Requires '`type`=radio' or '`type`=check'.

Used to determine whether a substring match of the value will cause a radio box or check box to be selected. If true, the match will happen whether the value is on a word boundary or not −− if false, the value must be on a word boundary. (When we speak of a word boundary, it is in the Perl sense −− a word character [A−Za−z0−9_] followed or preceded by a non−word character, or beginning or end of the string.)

#### 2.1.2.23. joiner

Requires 'type=links'.

With `type=links`, the accessories tag returns a link for each option. This allows you to override the default string ('<BR>') that joins these links. You can use Perl's metacharacter escapes, such as '\n' for newline or '\t' for tab.

#### 2.1.2.24. href

Requires 'type=links'.

This sets the base HREF for the link in a `links` type. Default is the current page.

#### 2.1.2.25. template

Requires 'type=links'.

Allows you to override the standard Interchange template for a hyperlink. You probably don't need to use this −− grep the code to grok it if you do (see 'sub build_accessory_links').

#### 2.1.2.26. form

Requires 'type=links'.

This sets the 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 −− one per item attribute value passed −− that set the variable "color" to the corresponding `passed` value (blank, blue, or green), do this:

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

This will generate something like the following:

```
<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.

#### 2.1.2.27. empty

Requires 'type=links'.

Setting '`empty=1`' includes a hyperlink for the empty "−−select−−" option. See the example in `form` above; if `empty=1` had been specified, three links would have been generated.

#### 2.1.2.28. secure

Requires 'type=links'.

Setting `secure`=1 causes the generated link(s) to point to your secure Interchange URL.

#### 2.1.2.29. new

Requires 'type=combo' or 'reverse_combo'.

You can use this to set a value in place of the 'New' or 'Current' option in a combo box. For example, if item 'os28044' has size attribute values of "Sm=10oz, Med=15oz, Lg=20oz":

```
  [accessories code=os28044 attribute=size type=combo new="my_new_value"]
------------------------------------------------------
  <INPUT TYPE=text NAME="mv_order_size" SIZE=16 VALUE="">
  <SELECT NAME="mv_order_size" SIZE="1">
  <OPTION VALUE="my_new_value">my_new_value
  <OPTION VALUE="Sm">10oz
  <OPTION VALUE="Med">15oz
  <OPTION VALUE="Lg">20oz</SELECT>
```

Or, with the default new value:

```
  [accessories code=os28044 attribute=size type=combo]
------------------------------------------------------
  <INPUT TYPE=text NAME="mv_order_size" SIZE=16 VALUE="">
  <SELECT NAME="mv_order_size" SIZE="1">
  <OPTION VALUE="">&lt;-- New
  <OPTION VALUE="Sm">10oz
  <OPTION VALUE="Med">15oz
  <OPTION VALUE="Lg">20oz</SELECT>
```

Default is no VALUE with option text set to '<-- New' for a combo box or 'Current -->' for a reverse_combo box.

# 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 positonal:

**ASP-like Perl call:**

Not applicable. The [and ...] tag only is used with [if ...], and Perl logic obviates the [if ...] tag.

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]
[area href=dir/page arg=page_arguments other_named_attributes]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| href | Path to Interchange page or action<br><br>*Special arguments*<br><br>♦ 'scan' links to a search (using search arguments in **arg**)<br>♦ 'http://...' external link (requires **form** attribute) | process |
| arg | Interchange arguments to page or action | *none* |

| *Attributes* | *Default* |
|---|---|
| form | *none* |
| search | *No* |
| secure | *No* |
| interpolate (reparse) | *No* |

| *Other_Charactreristics* | |
|---|---|

| | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* |

**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. assign

Allows you to assign numeric values to preempt calculation of one or more of the following tags:

- [handling], [salestax], [shipping], and [subtotal]

The assignment is persistent within a user's session until you clear it, an assigned tag will return your value instead of calculating a value.

Warning −− please be sure you understand the dependencies within the pricing system before using the `assign` tag.

## 2.4.1. Summary

```
[assign tag_name=value tag_name=value ...]
[assign clear=1]
```

| *Attributes* | *Description* | *Default* |
|---|---|---|
| clear | Clears all pending '`assign`' tag assignments | *none* |
| handling | Assigns an override value for [handling] tags | *none* |
| salestax | Assigns an override value for [salestax] tags | *none* |
| shipping | Assigns an override value for [shipping] tags | *none* |
| subtotal | Assigns an override value for [subtotal] tags | *none* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Container tag | *No* |

**ASP−like Perl call:**

```
$Tag->assign(  { shipping => 2.99, }  );
```

### 2.4.1.1. See Also

[handling], [salestax], [shipping], [subtotal], [Shipping]

## 2.4.2. Description

The `assign` tag allows you to assign numeric override values to one or more of the following tags:

- [handling], [salestax], [shipping], and [subtotal]

An assigned tag will return your value rather than calculating its own until you clear the assignment.

Assignment is persistent within the user's session (unless cleared) and affects only that user.

Assigning an empty string clears the tag's assignment. You can also clear all pending assignments at once with the `clear` attribute.

For example, the following eliminates salestax and sets shipping to $4.99 regardless of weight and destination:

```
[assign salestax=0 shipping=4.99]
```

This restores the [salestax] tag and eliminates handling charges:

```
[assign salestax="" handling=0]
```

This restores the normal behavior to the [shipping] and [handling] tags:

```
[assign clear=1]
```

Assignment affects only the value returned by a tag. Other behavior, such as formatting for the local currency, is not affected by the assignment.

Note −− you will get an error in the error log (and any pending assignment for the specified tag will be cleared) if you try to assign a value other than a number or the empty string ("").

#### 2.4.2.1. clear

Setting this to a true value clears all pending assignments (i.e., all assignable tags return to normal behavior).

#### 2.4.2.2. shipping

This sets the total value of shipping, rounded to locale−specific fractional digits. Always active if assigned a numeric value. See the [shipping] tag for detail about rounding, etc.

#### 2.4.2.3. handling

This option sets the total value of handling, rounded to fractional digits.

---

***Important note***

The [handling] tag is unlike the others in that it will be inactive (despite your assignment) unless the [value **mv_handling**] variable is true (i.e., a nonzero, non−blank value).

---

#### 2.4.2.4. salestax

This preempts the salestax calculation. The assigned value is not rounded.

#### 2.4.2.5. subtotal

This preempts the cart subtotal derived from prices. The assigned value is not rounded.

Note that you cannot assign to [total−cost] −− it will always be the sum of the four above.

Before using the `assign` tag, please be sure you understand the dependencies within the pricing system, such as the relationship between [total−cost] and assigned tags.

## 2.5. attr_list

## 2.5.1. 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−like Perl call:**

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

 OR

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

## 2.5.2. 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}
        {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:

#### 2.5.2.1. {key}

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

#### 2.5.2.2. {key|fallback string}

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

#### 2.5.2.3. {key true string}

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

#### 2.5.2.4. {key?} true text {/key?}

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

#### 2.5.2.5. {key:} false text {/key:}

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

## 2.6. banner

Implements random or rotating banner ads. See also Banner/Ad rotation.

### 2.6.1. Summary

```
[banner category]
[banner category=my_category other_named_attributes]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| category | | default |

| *Attributes* | *Default* |
|---|---|
| table | banner |
| r_field (unweighted) | rotate |
| b_field | banner |
| separator (unweighted) | ':' |
| delimiter (unweighted) | '{or}' |
| weighted | *No* |
| once (weighted) | *No* |
| c_field (weighted) | category |
| w_field (weighted) | weight |
| interpolate (reparse) | *No* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Container tag | *No* |

**Tag expansion example:**

```
[banner category=]
```

**ASP−like Perl call:**

```
$Tag->banner(  { category => $key, } );
```

or similarly with positional parameters,

```
$Tag->banner($category, $attribute_hash_reference);
```

#### 2.6.1.1. See Also

Banner/Ad rotation

## 2.6.2. Description

Implements random or rotating banner ads. See Banner/Ad rotation in the Interchange Template documentation.

You will need a banner ad table (typically called 'banner') which contains banner data. The following is an example:

| *code* | *category* | *weight* | *rotate* | *banner* |
|---|---|---|---|---|
| m_3 | cat1 | 7 | 0 | my banner 3 |
| m_1 | cat1 | 1 | 0 | my banner 1 |
| default | | | 1 | Default 1{or}Default 2{or}Default 3 |
| m_2 | cat1 | 2 | 0 | my banner 2 |
| t_1 | cat2 | 4 | 0 | their banner 1 |
| t_2 | cat2 | 1 | 0 | their banner 2 |

#### 2.6.2.1. category

Default: category="default"

This specifies the category for weighted ad, or the table row (i.e., code value) for an unweighted ad.

#### 2.6.2.2. table

Default: table="banner"

Setting '`table`="my_banner_table"' forces the tag to refer to 'my_banner_table' rather than the default 'banner' table for banner ad information.

#### 2.6.2.3. r_field

Default: r_field="rotate"

Unweighted ads only.

A table row may include multiple banners in the 'banner' column. The column specified by r_field contains a boolean that determines whether to rotate banners. In the above table example, 'Default 1', 'Default 2' and 'Default 3' would rotate.

#### 2.6.2.4. b_field

Default: b_field="banner"

This specifies the column containing the banner descriptor(s). The default is 'banner'. Note that an entry might be a delimited list of banner descriptors to rotate (see delimiter below).

#### 2.6.2.5. separator

Default: separator=":"

Unweighted ads only.

This sets the separator within the table key (i.e., code) for multi−level categorized ads. See Banner/Ad rotation.

#### 2.6.2.6. delimiter

Default: delimiter="{or}"

Unweighted ads only.

This specifies the delimiter between rotating banner descriptors in the 'banner' column.

#### 2.6.2.7. weighted

The `banner` tag will not apply weighting from the table unless you set `weighted`=1. Note that the tag will behave as if you gave it a standard unweighted entry −− it will look for a matching row rather than a matching category.

#### 2.6.2.8. once

Weighted ads only.

If the option once is passed (i.e., [banner once=1 weighted=1], then the banners will not be rebuilt until the total_weight file is removed. See Banner/Ad rotation.

#### 2.6.2.9. c_field

Default: c_field="category"

Weighted ads only.

This specifies the column containing the banner category for weighted ads. The `banner` tag will display ads from rows in the table whose category matches the category given in the tag, with frequency corresponding to the weights in the table.

#### 2.6.2.10. w_field

Default: w_field="weight"

Weighted ads only.

This specifies the column containing the banner weight.

# 2.7. bounce

## 2.7.1. 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−like Perl call:**

None. This tag doesn't work with embedded Perl due to special processing.

## 2.7.2. 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]
```

# 2.8. calc

Calculates the value of the enclosed arithmetic expression.

## 2.8.1. Summary

```
[calc] Expression [/calc]
```

**No parameters**

**No attributes** (though you can break it if you set '`interpolate=0`')

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Has Subtags | *No* |
| Container tag | *Yes* |
| Nests | *No* |

**ASP−like Perl call:**

There is never a reason to call this tag from within perl or ASP code. Simply do the calculation directly.

## 2.8.2. Description

Calculates the value of the enclosed arithmetic expression.

Use it as follows: [calc] *Expression* [/calc]

The enclosed region where the arguments are calculated according to normal arithmetic symbols. For instance:

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

will expand to:

```
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.

# 2.9. cart

## 2.9.1. 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−like Perl call:**

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

OR

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

## 2.9.2. Description

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

# 2.10. catch

The page content contained within the [`catch` *label*][`/catch`] block executes if the correspondingly labelled `try` block fails.

You can also return a result based on the error message caught in the try block with paired subtags, like this:

```
   [error message]body text[/error message]
```

Note that this feature excises *all* tag/endtag pairs if interpolation is turned off, so the `catch` tag interpolates by default.

See also [`try`].

## 2.10.1. Summary

```
[try my_label]
    Body text to return if no error
[/try]
.
.
.
[catch label=my_label other_named_attributes]
    [/Pattern matching error message 1/]
        Return this if error 1 occurs
    [/Pattern matching error message 1/]

    [/Pattern matching error message 2/]
        Return this if error 2 occurs, etc.
    [/Pattern matching error message 2/]

    Default body text to process if try block caused an error
[/catch]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| label | The label shared by the paired `try` and `catch` blocks | 'default' |

| *Attributes* | *Default* |
|---|---|
| interpolate | *Yes* |
| reparse | *Yes* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Container tag | Yes |
| Has Subtags | [*Error message text*]<br><br>body<br>[*/Error message text*] |

**Tag expansion example**

Ignoring whitespace, the following would return division result if successful, 0 on a division by zero, or an error message:

```
   [set divisor]0[/set]
   [try label=div]
      [perl] eval(1 / [scratch divisor]) [/perl]
   [/try]
   [catch div]
      [/Illegal division by zero/]
         0
      [/Illegal division by zero/]
      [/eval "string" trapped by operation mask/]
         Perl Safe error
      [/eval "string" trapped by operation mask/]
      Other division error
   [/catch]
---
```

```
Perl Safe error
```

**ASP-like Perl call:**

```
$Tag->catch(  { label => I<'my_label'>, },
              $body  );
```

or similarly with positional parameters,

```
$Tag->catch($label, $attribute_hash_reference, $body);
```

#### 2.10.1.1. See Also

try

## 2.10.2. Description

The page content contained within the [`catch` *label*][`/catch`] block executes if the correspondingly labelled `try` block fails. The catch block executes in place on the page if triggered (*i.e.*, it does not return its result in place of the try block).

You can also return a result based on the error message caught in the try block with paired subtags, like this:

```
[/error message/]special catch block for the error[/error message/]
```

The error message to use in the special block will generally be part of the entry the error generates in your error log. For example, a division by zero error generates something like the following in the error log:

```
127.0.0.1 4cU3Pgsh:127.0.0.1 - [24/May/2001:14:45:07 -0400]\
tag /cgi-bin/tag72/tag Safe: Illegal division by zero\
at (eval 526) line 2.
```

(note that trailing backslashes in the example indicate a continued line).

#### 2.10.2.1. label

This is the label specifying the corresponding [`try` block. Defaults to 'default'.

# 2.11. cgi

## 2.11.1. 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−like Perl call:**

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

 OR

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

## 2.11.2. 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.

# 2.12. checked

## 2.12.1. 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−like Perl call:**

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

OR

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

## 2.12.2. 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`.

# 2.13. control

Returns named scratchpad field or copies named scratch variable to scratchpad. Returns value specified by 'default' attribute if scratchpad variable is undefined or empty. Calling without a name moves to next scratchpad. Calling without a name and 'reset=1' returns to first scratchpad page.

## 2.13.1. Summary

Parameters: **name default**

- name
  - Name of scratchpad variable to set or return
- default
  - Value to return if scratchpad variable missing or empty

Attributes

- reset (must not specify name; may specify default)
  - Resets scratchpad (i.e. $::Control array) by setting special scratch variable 'control_index' to 0. Control_index is an index into the $::Control == $Vend::Session−>{control} array of hashrefs.

- set
    - Copies named scratch variable (i.e., from $::Scratch) to scratchpad with current control index.

**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−like Perl call:**

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

OR

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

### 2.13.2. Description

Returns named scratchpad field or copies named scratch variable to scratchpad. Returns value specified by 'default' attribute if scratchpad variable is undefined or empty. Calling without a name moves to next scratchpad. Calling without a name and 'reset=1' returns to first scratchpad page.

## 2.14. control_set

Bulk−sets scratchpad variables on the scratchpad page specified by 'index'. Note that, unlike [control], this does not copy values from scratch.

### 2.14.1. Summary

This example sets var_one, var_two and var_three in the scratchpad on page 5 (index begins with 0).

```
  [control_set index=4]
    [var_one]var_one_value[/var_one]
    [var_two]var_two_value[/var_two]
    [var_three]var_three_value[/var_three]
  [/control_set]
```

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−like Perl call:**

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

 OR

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

## 2.14.2. Description

Bulk−sets scratchpad variables on the scratchpad page specified by 'index'. Note that, unlike [control], this does not copy values from scratch.

# 2.15. counter

## 2.15.1. Summary

Parameters: **file**

Positional parameters in same order.

Invalidates cache: **YES**

Called Routine:

**ASP−like Perl call:**

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

 OR

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

Attribute aliases

```
            name ==> file
```

## 2.15.2. 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:

#### 2.15.2.1. decrement=1

Causes the counter to count down instead of up.

#### 2.15.2.2. start=50

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

#### 2.15.2.3. value=1

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

# 2.16. currency

## 2.16.1. 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−like Perl call:**

```
$Tag−>currency(
    {
     convert => VALUE,
```

```
         noformat => VALUE,
        },
        BODY
    )

OR

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

## 2.16.2. 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.

# 2.17. data

## 2.17.1. 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−like Perl call:**

```
    $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
```

## 2.17.2. 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 [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.

# 2.18. default

## 2.18.1. 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−like Perl call:**

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

OR

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

## 2.18.2. 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.

# 2.19. description

## 2.19.1. 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−like Perl call:**

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

 OR

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

## 2.19.2. 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.

# 2.20. discount

## 2.20.1. 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−like Perl call:**

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

 OR
```

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

## 2.20.2. 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]
```

# 2.21. dump

Dumps client connection information, cart contents, query value, contents of environment, session, and CGI with Data::Dumper to the page. This is useful for debugging.

## 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-like Perl call:**

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

OR

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

## 2.21.2. Description

Dumps client connection information, cart contents, query value, contents of environment, session, and CGI with Data::Dumper to the page. This is useful for debugging.

# 2.22. ecml

Uses ECML (Electronic Commerce Markup Language) module to map Interchange forms/userdb to ECML checkout

## 2.22.1. Summary

Parameters: **name function**

- function (default = 'widget')

```
Place form boxes on page:

         [ecml name]
         [ecml address]

Magic database entry from country database:

        [ecml country]

Map values back to Interchange variables for saving in UserDB:

        <INPUT TYPE=hidden NAME=mv_click CHECKED VALUE="ECML_map">
        [set ECML_map]
        [ecml function=mapback]
        [/set]
```

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−like Perl call:**

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

 OR

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

## 2.22.2. Description

This package implements the ECML standard for the Interchange demo. ECML stands for "Electronic Commerce Modeling Language", but at this writing it is a simple standard for naming variables so that "electronic wallets" can pre−fill−in your checkout form based on users past purchase from other companies.

It translates into ECML from the following Interchange variables:

| *ECML* | *Interchange variable* |
|---|---|
| Ecom_BillTo_Online_Email | b_email |
| Ecom_BillTo_Postal_City | b_city |
| Ecom_BillTo_Postal_CountryCode | b_country |
| Ecom_BillTo_Postal_Name_First | b_fname |
| Ecom_BillTo_Postal_Name_Last | b_lname |
| Ecom_BillTo_Postal_Name_Middle | b_mname |
| Ecom_BillTo_Postal_Name_Prefix | b_title |
| Ecom_BillTo_Postal_Name_Suffix | b_name_suffix |
| Ecom_BillTo_Postal_PostalCode | b_zip |
| Ecom_BillTo_Postal_StateProv | b_state |
| Ecom_BillTo_Postal_Street_Line1 | b_address1 |
| Ecom_BillTo_Postal_Street_Line2 | b_address2 |
| Ecom_BillTo_Postal_Street_Line3 | b_address3 |
| Ecom_BillTo_Telecom_Phone_Number | b_phone_day |
| Ecom_ConsumerOrderID | mv_order_number |
| Ecom_Payment_Card_ExpDate_Day | mv_credit_card_exp_day |
| Ecom_Payment_Card_ExpDate_Month | mv_credit_card_exp_month |
| Ecom_Payment_Card_ExpDate_Year | mv_credit_card_exp_year |
| Ecom_Payment_Card_Name | c_name |
| Ecom_Payment_Card_Number | mv_credit_card_number |
| Ecom_Payment_Card_Protocol | payment_protocols_available |
| Ecom_Payment_Card_Type | mv_credit_card_type |
| Ecom_Payment_Card_Verification | mv_credit_card_verify |
| Ecom_ReceiptTo_Online_Email | r_email |
| Ecom_ReceiptTo_Postal_City | r_city |
| Ecom_ReceiptTo_Postal_CountryCode | r_country |
| Ecom_ReceiptTo_Postal_Name_First | r_fname |
| Ecom_ReceiptTo_Postal_Name_Last | r_lname |
| Ecom_ReceiptTo_Postal_Name_Middle | r_mname |
| Ecom_ReceiptTo_Postal_Name_Prefix | r_title |
| Ecom_ReceiptTo_Postal_Name_Suffix | r_name_suffix |
| Ecom_ReceiptTo_Postal_PostalCode | r_zip |
| Ecom_ReceiptTo_Postal_StateProv | r_state |
| Ecom_ReceiptTo_Postal_Street_Line1 | r_address1 |
| Ecom_ReceiptTo_Postal_Street_Line2 | r_address2 |
| Ecom_ReceiptTo_Postal_Street_Line3 | r_address3 |
| Ecom_ReceiptTo_Telecom_Phone_Number | r_phone |

| | |
|---|---|
| Ecom_SchemaVersion | ecml_version |
| Ecom_ShipTo_Online_Email | email |
| Ecom_ShipTo_Postal_City | city |
| Ecom_ShipTo_Postal_CountryCode | country |
| Ecom_ShipTo_Postal_Name_Combined | name |
| Ecom_ShipTo_Postal_Name_First | fname |
| Ecom_ShipTo_Postal_Name_Last | lname |
| Ecom_ShipTo_Postal_Name_Middle | mname |
| Ecom_ShipTo_Postal_Name_Prefix | title |
| Ecom_ShipTo_Postal_Name_Suffix | name_suffix |
| Ecom_ShipTo_Postal_PostalCode | zip |
| Ecom_ShipTo_Postal_StateProv | state |
| Ecom_ShipTo_Postal_Street_Line1 | address1 |
| Ecom_ShipTo_Postal_Street_Line2 | address2 |
| Ecom_ShipTo_Postal_Street_Line3 | address3 |
| Ecom_ShipTo_Telecom_Phone_Number | phone |
| Ecom_TransactionComplete | end_transaction_flag |

Once the form variables are input and sent to Interchange, the [ecml function=mapback] tag will cause the input results to be mapped back from the ECML names to the Interchange names.

If you only have a `name` variable in your UserDB, the module will attempt to split it into first name and last name for ECML purposes and map the results back. If you have `fname` and `lname`, then it will not.

# 2.23. 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.

## 2.23.1. Summary

```
[either]
  This
[or]
  That
[or]
  The other
[/either]
```

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−like Perl call:**

```
    $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−like Perl call:**

```
    $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 refered 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 formating can bet 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

Exports a database to a delimited text file (see also import).

### 2.25.1. Summary

Parameters: **table**

Positional parameters in same order.

- table
  - ♦ The table to export
- file
- Filename to export to. Note that the NoAbsolute directive and other conditions may affect actual location of the output file.
- type
  - ♦ Specifies the [line, record] delimiter types. Either NOTES or one of the following:

```
my %Delimiter = (
      2 => ["\n", "\n\n"],
      3 => ["\n%%\n", "\n%%%\n"],
      4 => ["CSV","\n"],
```

```
    5 => ['|', "\n"],
    6 => ["\t", "\n"],
    7 => ["\t", "\n"],
    8 => ["\t", "\n"],
    LINE => ["\n", "\n\n"],
    '%%%' => ["\n%%\n", "\n%%%\n"],
    '%%' => ["\n%%\n", "\n%%%\n"],
    CSV => ["CSV","\n"],
    PIPE => ['|', "\n"],
    TAB => ["\t", "\n"],
    );
```

- If using NOTES
  - notes_separator (defaults to "\f")
  - notes_field (defaults to "notes_field")

- field
  - The column to add (or delete if delete and verify are true)
- delete
  - If 'verify' attribute also set, deletes column specified by 'field' attribute rather than adding a column.
- verify
  - must be true when deleting a column
- sort
  - Output sorted rows (usage: sort="*sort_field*:*sort_option*") (see search/form variable 'mv_sort_option' for sort options)

**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−like Perl call:**

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

OR

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

Attribute aliases

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

### 2.25.2. Description

Exports 'table' to a delimited text file. See also `import` tag which imports files into databases.

## 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−like Perl call:**

```
    $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−like Perl call:**

```
    $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−like Perl call:**

```
    $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 `&quot;`, 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 {
                        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;
                $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 a 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

Controls Interchange flags. For example, flags affect database access and transactions for those databases able to support these features. See also the [`tag`] tag.

## 2.29.1. Summary

Parameters: **type**

type may be one of the following

- read
  - Flags the table read−only
- write
  - Flags the table writeable by default (or read−only if you also set the value=0 attribute)
- transactions
  - Reopens the database in transactions mode if Safe.pm is not active (e.g., in a global subroutine, usertag or [perl global=1] tag). The limitation exists because it is not possible to reopen a database within Safe.pm.
- commit
  - Attempts to commit transactions
- rollback
  - Attempts to rollback transactions
- build
  - Forces build of static Interchange page specified by the name attribute
- checkhtml

## 2.29.2. Attributes

- 'flag' and 'name'
  - Aliases for 'type' (except for 'type=build')
- tables
  - The name of the table to flag
  - 'table' is an alias
- value
  - The boolean value of the flag
- name
  - Name of page to build or alias for 'type'
- show
  - Normally, the [flag] tag returns nothing to the page. Setting 'show=1' causes the tag to return status, if any.

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

**ASP−like Perl call:**

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

 OR

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

Attribute aliases

```
           flag ==> type
           name ==> type
          table ==> tables
```

### 2.29.3. Description

The `flag` tag controls database access and transactions.

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

For example, you can call

```
[flag type=write value=1 table=products]
```

to mark the `products` DBM database writable. **This must be done before ANY access to that table.**

Note that SQL databases are always writable if allowed by the SQL database itself, and in−memory databases will never be written.

Using [`flag` build] forces static build of a page, even if it contains dynamic elements.

## 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−like Perl call:**

```
    $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−like Perl call:**

```
$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

Skips page content between [`goto` *name*] and [`label` *name*]. Note that the `goto` tag is not interpreted in the standard way, and you cannot use the '$Tag−>goto()' Perl syntax. Note also that skipping endtags with `goto` will probably break your page.

### 2.32.1. Summary

```
[goto name=label_name if=condition]
  content to skip
[label name=label_name]
```

or positionally,

```
[goto name if]
  content to skip
[label name]
```

| ***Parameters*** | ***Description*** | ***Default*** |
|---|---|---|
| name | The name set in the corresponding [label] tag | *none* |
| if | Condition for `goto`. Should evaluate to truth value before tag is parsed. | *true* |

| ***Other_Charactreristics*** | |
|---|---|
| Container tag | *No*, but you use it like this:<br><br>[goto<br>name=*label_name*<br>if=*condition*]<br><br>*body text*<br>[label *label_name*] |
| Has Subtags | [label] interpreted by `goto` |

**ASP−like Perl call:**

No Perl call available (Note that this tag is not parsed in the standard way).

### 2.32.2. Description

Skips page content between [`goto` *name*] and [`label` *name*]. Note that the `goto` tag is not interpreted in the standard way, and you cannot use the '$Tag−>goto()' Perl syntax. Note also that skipping endtags with `goto` will probably break your page.

The correspondingly named [`label`] tag marks the end of the page content the `goto` should skip. Note that the [`label`] tag is not an end tag, but simply a marker for the end of the text to skip.

***Technical note (Interchange 4.8):*** This tag may not work properly if you have more than one goto/label pair on a page.

#### 2.32.2.1. name

This should match the name set in a [label] tag *after* the goto tag in the page (i.e., don't create loops).

#### 2.32.2.2. if

Condition for goto. If the argument to '`if`' is true, the tag will skip the text between the `goto` and <label>. Note that the tag itself does not evaluate the condition. The condition must evaluate to a true or false value before the `goto` tag processes it.

For example, this will not execute the goto:

```
[set go]0[/set]
[goto name="there" if="[scratch go]"]
```

# 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−like Perl call:**

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

 OR

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

Attribute aliases

```
            carts ==> 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

Test harness block. Similar to try/catch. Interprets the body text and checks the return value against expected and explicitly bad cases.

Returns DIED, OK, or NOT OK message along with your result if not the expected value.

### 2.34.1. Summary

```
[harness expected="good" name=my_test_number_1]
  [good]The Expected Return Value[/good]
  [not]Some Specifically Bad Return Value[/not]
  Tags and code to test here
[/harness]
```

No parameters.

- expected (default "OK")
  - Tagname for delimiting your expected return value
- name (default "test*nnn*")
  - This will appear in your output message (useful for distinguishing harness tags from one another)

**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−like Perl call:**

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

 OR

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

### 2.34.2. Description

Test harness block. Similar to try/catch. Interprets the body text and checks the return value against expected and explicitly bad cases.

Returns DIED, OK, or NOT OK message along with the harness name and your result if not the expected value.

## 2.35. href

Alias for [area] tag.

## 2.36. html_table

Builds an HTML table

### 2.36.1. Summary

- columns
  - Whitespace−delimited list of columns
- delimiter (default "\t")
  - Line delimiter to use if tag body is delimited text rather than an array reference
- record_delim (default "\n")
  - Record delimiter to use if tag body is delimited text rather than an array reference
- tr
  - HTML attributes for <TR>
- td
  - HTML attributes for <TD>
- th
  - HTML attributes for <TH>
- fc
  - HTML attributes for <TD> in the first cell
- fr
  - HTML attributes for <TR> in the first row

**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−like Perl call:**

```
$Tag->html_table(
```

```
        {
        },
        BODY
    )

OR

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

## 2.36.2. Description

Builds an HTML table

# 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 positonal:

**ASP-like Perl call:**

Not applicable. Any [if ...] call can be better and more efficiently done with Perl.

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
                            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]';
    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. pragma

The Interchange Pragma settings, set with the the catalog.cfg manpage directive `Pragma` or with `[pragma name]`.

```
[if pragma dynamic_variables]
__THE_VARIABLE__
[else]
[data table=variable column=Variable key=THE_VARIABLE]
[/else]
[/if]
```

#### 2.37.2.9. 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]
```

#### 2.37.2.10. session

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

#### 2.37.2.11. 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.

#### 2.37.2.12. 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].

#### 2.37.2.13. [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.

#### 2.37.2.14. [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.

#### 2.37.2.15. [else] text [/else]

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

#### 2.37.2.16. [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 [*per*l] 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−like Perl call:**

```
    $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-like Perl call:**

Not applicable.

## 2.39.2. Description

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

# 2.40. index

Creates an index for the specified table.

## 2.40.1. Summary

Parameters: **table**

Positional parameters in same order.

- extension (default "idx")
  - ♦ Index file extension
- basefile
  - ♦ Database filename. Exports the table to this filename if old or missing before indexing. See also the `export` tag for additional relevent attributes such as delimiter type, etc.
- export_only
  - ♦ Just do the export if necessary (not the index).
- spec
  - ♦ The index specification
- fn or fields or col or columns
  - ♦ field(s) to index
- show_status
  - ♦ Return '1' to the page if successful

**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−like Perl call:**

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

 OR

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

Attribute aliases

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

### 2.40.2. Description

Creates an index for the specified table.

## 2.41. item_list

### 2.41.1. 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−like Perl call:**

```
    NOTE: This would not usually be used with embedded Perl −− a better
    choice would normally be:

                 for(@$Items) { CODE }

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

 OR

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

Attribute aliases

```
            cart ==> name
```

### 2.41.2. 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:

#### 2.41.2.1. [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.

#### 2.41.2.2. [if−data ! table column]

Reverses sense for [if−data].

#### 2.41.2.3. [/if−data]

Terminates an [if_data table column] element.

#### 2.41.2.4. [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.

#### 2.41.2.5. [if−field ! fieldname]

Reverses sense for [if−field].

#### 2.41.2.6. [/if−field]

Terminates an [if_field fieldname] element.

#### 2.41.2.7. [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.

#### 2.41.2.8. [item−code]

Evaluates to the product code for the current item.

#### 2.41.2.9. [item−data database fieldname]

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

#### 2.41.2.10. [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.

#### 2.41.2.11. [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.

#### 2.41.2.12. [item−increment]

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

#### 2.41.2.13. [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.

#### 2.41.2.14. [item−modifier attribute]

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

#### 2.41.2.15. [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.)

#### 2.41.2.16. [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.

#### 2.41.2.17. [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.

#### 2.41.2.18. [item−discount]

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

#### 2.41.2.19. [item−discount_subtotal]

Inserts the discounted subtotal of the ordered items.

#### 2.41.2.20. [item−quantity]

Evaluates to the quantity ordered for the current item.

#### 2.41.2.21. [item−subtotal]

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

#### 2.41.2.22. [modifier−name attribute]

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

#### 2.41.2.23. [quantity−name]

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

## 2.42. label

The page label for goto. See [goto] tag for description. Note that this is not a standard tag, but is simply a marker used by goto.

Parameter: **name**

```
[goto name=label_name if=condition]
  content to skip
[label name=label_name]
```

## 2.43. log

Log contained text to specified file.

### 2.43.1. Summary

Parameters: **file**

- file

        - name of file to log to. 'file=">*filename*"' also sets 'create' attribute.
    - create
        - Set create=1 to create the file if not present
    - process
        - Processing (if any) to apply to the content while logging
            - nostrip (don't strip leading/trailing whitespace and convert "\r\n" to "\n"
    - delim and record_delim
        - Line and record delimiters, respectively
    - type
        - Log type
            - text (ordinary text file)
            - quot (delimited entries)
            - error (add Interchange error formatting and time/location stamps)
    - hide
        - Suppress status otherwise returned by tag to the page.

**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−like Perl call:**

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

OR

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

Attribute aliases

```
            arg ==> file
```

### 2.43.2. Description

Log contained text to specified file.

## 2.44. loop

## 2.44.1. 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−like Perl call:**

```
    NOTE: This would not usually be used with embedded Perl −− a better
    choice would normally be:

                 for(@list) { CODE }

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

 OR

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

Attribute aliases

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

## 2.44.2. 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 list="A B C"]
```

```
    [loop prefix=mid list="[loop-code]1 [loop-code]2 [loop-code]3"]
        [loop prefix=inner list="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 world "Americana", and will work the same in both old and new syntax.

Ranges are accepted when you pass a `list` if you set the `ranges` option:

```
[loop list="A..Z" ranges=1][loop-code] [/loop]
```

The above lists all of the characters from A to Z. Any Perl incrementing variable list will work, but most commonly a range would be something like `1..100`. You can mix regular sets -- `1..5 10 20` would produce the list `1 2 3 4 5 10 20`.

If you surround the repeating text section with a `[list] [/list]` anchor, the more-list, ml=N, and on-match / no-match processing is done just as in `[query]` and `[search-region]`.

Using the `acclist` option will parse Interchange option lists, as used in product options. The value is available with `[loop-code]`, the label with `[loop-param label]`. If the size data for SKU `TS-007` was set to the string `S=Small, M=Medium, L=Large, XL=Extra Large` then you could produce a select list of options this way:

```
[loop list="[data products size TS-007]" acclist=1]
        [on-match]<SELECT NAME=mv_order_size>[/on-match]
            [list]<OPTION VALUE="[loop-code]"> [loop-param label]</OPTION>[/list]
    [on-match]</SELECT>[/on-match]
[/loop]
```

Of course the above is probably more easily produced with `[accessories code=TS-007 attribute=size]`, but there will be other uses for the capability. For instance:

```
     <SELECT NAME=Season>
[loop acclist=1
            list="
                    Q1=Winter,
                    Q2=Spring,
                    Q3=Summer,
                    Q4=Fall
            "]> <OPTION VALUE="[loop-code]"> [loop-param label]</OPTION>
[/loop]
```

If your parameter list needs to have spaces in the parameters, surround them with double or single quotes and set the quoted=1 option: in product options. If the size data for SKU `TS-007` was set to the string `S=Small, M=Medium, L=Large, XL=Extra Large` then you could produce a select list of options this way:

```
[loop list="[data products size TS-007]" acclist=1]
        [on-match]<SELECT NAME=mv_order_size>[/on-match]
            [list]<OPTION VALUE="[loop-code]"> [loop-param label]</OPTION>[/list]
    [on-match]</SELECT>[/on-match]
[/loop]
```

#### 2.44.2.1. [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].

#### 2.44.2.2. [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].

#### 2.44.2.3. [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].

#### 2.44.2.4. [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 parmeter `N` is non-empty, and the **ELSE** (if any) otherwise.

See [if-PREFIX-pos].

#### 2.44.2.5. [loop-accessories]

Outputs an [accessories ...] item.

See [PREFIX-accessories].

#### 2.44.2.6. [loop-change marker]

See [PREFIX-change].

#### 2.44.2.7. [loop-code]

Evaluates to the code for the current item.

See [PREFIX-code].

#### 2.44.2.8. [loop−data database fieldname]

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

See [PREFIX−data].

#### 2.44.2.9. [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].

#### 2.44.2.10. [loop−field fieldname]

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

See [PREFIX−field].

#### 2.44.2.11. [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].

#### 2.44.2.12. [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.

#### 2.44.2.13. [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.

#### 2.44.2.14. [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.

# 2.45. mail

Mail contained text to recipient specified by 'to' using the program specified with the SendMailProgram catalog directive.

## 2.45.1. Summary

Parameters: **to**

Positional parameters in same order.

- raw
  - ♦ Send it raw without creating headers and checking content, recipient, subject, etc.
- extra
  - ♦ Additional headers (these will also be added to 'raw' messages)
- success
  - ♦ Tag return value if successful (default is 1).
- hide
  - ♦ Suppress tag return value. This would otherwise be the 'success' attribute setting.
- show
  - ♦ The tag will return the final message with headers in the page

**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−like Perl call:**

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

 OR
```

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

### 2.45.2. Description

Mail contained text to recipient specified by 'to' using the program specified with the SendMailProgram catalog directive.

## 2.46. mvasp

Executes the ASP−style perl code contained by the tag. The code will run under the restrictions of the Safe module. This is very similar to the [perl] tag, except that the standard '<%' and '%>' ASP delimiters allow you to mix HTML and perl code.

### 2.46.1. Summary

```
[mvasp tables] ASP here [/mvasp]
[mvasp tables="db1 db2 ..." other_named_attributes] ASP here [/mvasp]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| tables | Database tables to be made available to ASP Perl code | *none* |
| table | Alias for `tables` | *none* |

| *Attributes* | *Default* |
|---|---|
| failure | *none* |
| no_return | *Always true* |
| subs | *No* |
| arg="subs" | *Same as* `subs` |
| global | *No* |
| file | *none* |
| interpolate | *No* |
| reparse | *No* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *Yes* |
| Has Subtags | *<% and %>* |
| Container tag | *Yes* |
| Nests | *No* |

**Tag expansion example:**

```
[mvasp tables="products" failure="ASP Broke <BR>"]
   <P>This is HTML</p>
   <% my $sku = $Values->{code}; %>
   <P>More HTML</p>
   <% my $result = "Looked up SKU $sku. It is a ";
      $result .= $Tag->data('products', 'description', $sku );
      $Document->write( "$result <br>\n" ); %>
   <P>Still more HTML</p>
[/mvasp]
```

```
-----------------------------------------------------
      <P>This is HTML</p>

      <P>More HTML</p>
      Looked up SKU os28044. It is a Framing Hammer <br>

      <P>Still more HTML</p>
```

#### 2.46.1.1. See Also

perl, Interchange Programming

## 2.46.2. Description

Executes the ASP−style perl code contained by the tag. The code will run under the restrictions of the Safe module. This is very similar to the [perl no_return=1] tag, except that the standard '<%' and '%>' ASP delimiters allow you to mix HTML and perl code.

See the perl tag and ASP−Like Perl sections for more detail.

#### 2.46.2.1. tables

Whitespace−separated list of database tables to make available within the ASP−Perl code. See perl tag.

#### 2.46.2.2. failure

The value the tag should return in case the perl code fails the eval. See perl tag.

#### 2.46.2.3. no_return

The return value of the perl code is always suppressed. If you want output from the ASP code sections, you must explicitly write it with the &HTML or $Document−>write() functions.

You can also retrieve the return value of the perl code from the session hash via `[data session mv_perl_result]`. See perl tag.

#### 2.46.2.4. subs

Enable GlobalSub routines (requires catalog directive AllowGlobal). See perl tag.

#### 2.46.2.5. global

Turn off Safe protection (requires catalog directive AllowGlobal). See perl tag.

#### 2.46.2.6. file

Prepend the contents of the specified file or FileDatabase entry to the perl code before `eval`'ing it. See perl tag.

#### 2.46.2.7. Examples

See the ASP−Like Perl section of Interchange Programming.

# 2.47. nitems

Returns the total number of items ordered. Uses the current cart if none specified.

## 2.47.1. Summary

Parameters: **name**

- name
  - Cart name
  - Default: current cart
- qualifier
  - An item attribute that must be true in order to count the item.
  - Default: None
- compare
  - Regular expression the specified qualifier attribute's value must match to be counted. This replaces the truth value comparison.
  - Default: None (uses truth value of the specified qualifier attribute)

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

**ASP−like Perl call:**

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

OR

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

## 2.47.2. Description

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

# 2.48. options

Builds HTML widgets as defined in the options table for selecting options associated with a given product. This tag handles simple, matrix or modular options. See also the `accessories` tag.

Here is an illustrative example from the 'tools' sample data set of the foundation catalog:

```
===
   [options code=os28005]
---
   <input type=hidden name=mv_item_option value="logo">
     <SELECT NAME="mv_order_logo">
     <OPTION VALUE="c">Construct Something
     <OPTION VALUE="y" SELECTED>Your Logo</SELECT><BR>
   <input type=hidden name=mv_item_option value="color">
     <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BLK" > Black
     <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BEIGE" > Beige
     <INPUT TYPE="radio" NAME="mv_order_color" VALUE="WHITE" > White<BR>
   <input type=hidden name=mv_item_option value="bristle">
     <SELECT NAME="mv_order_bristle">
     <OPTION VALUE="synthetic">Synthetic
     <OPTION VALUE="camel">Camel Hair</SELECT>
===
```

## 2.48.1. Summary

Parameters: **code**

- code
  - Product key (usually sku).
  - No default
- table
  - Table to use for option attributes.
  - Default: 'options'
- td
  - Results as table rows. For example, compare the following example from the 'tools' sample data set with the earlier example:

```
===
   [options code=os28005 td=1]
---
   <td><input type=hidden name=mv_item_option value="logo">
      <SELECT NAME="mv_order_logo">
      <OPTION VALUE="c">Construct Something
      <OPTION VALUE="y" SELECTED>Your Logo</SELECT></td>
   <td><input type=hidden name=mv_item_option value="color">
      <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BLK" > Black
      <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BEIGE" > Beige
      <INPUT TYPE="radio" NAME="mv_order_color" VALUE="WHITE" > White</td>
   <td><input type=hidden name=mv_item_option value="bristle">
      <SELECT NAME="mv_order_bristle">
      <OPTION VALUE="synthetic">Synthetic
      <OPTION VALUE="camel">Camel Hair</SELECT></td>
===
```

(Note that the output was reformatted to fit this page)

- price
  - Default: False
  - Boolean. If set and the options have prices, the HTML widget(s) will show the prices. This is like the `price` attribute of the accessories tag.
    Note that the `price data` setting comes from the 'price' column of the options table.
    Technical note−− If your options table has different mappings, you can control this with

$::Variable->{MV_OPTION_TABLE_MAP}

- label
  - Shows labels for the options with the widgets.
  - Default: False

The following example (using another item from the 'tools' data) illustrates the price and label attributes:

```
===
   [options code=os28011 label=1 price=1]
---
   Handle<BR>
     <input type=hidden name=mv_item_option value="handle">
       <SELECT NAME="mv_order_handle">
       <OPTION VALUE="W">Wood handle
       <OPTION VALUE="E">Ebony handle ($20.00)</SELECT><BR>
   Blade material<BR>
     <input type=hidden name=mv_item_option value="blade">
       <SELECT NAME="mv_order_blade">
       <OPTION VALUE="P">Plastic blade ($-1.22)
       <OPTION VALUE="S" SELECTED>Steel blade
       <OPTION VALUE="T">Titanium blade ($100.00)</SELECT>
===
```

(again, the output has been reformatted to fit the page).

- bold
  - Boldfaces the labels if the `'label'` option is set.
  - Default: False
- joiner
  - The joiner for the widgets.
  - Default: <BR>

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

**ASP-like Perl call:**

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

 OR

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

# 2.49. or

## 2.49.1. 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 positonal:

**ASP−like Perl call:**

```
    $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
```

## 2.49.2. Description

**NO Description**

# 2.50. order

Expands into a hypertext link which will include the specified item in the list of products to order and display the order page.

## 2.50.1. Summary

```
    [order code quantity]Link Text[/order]
    [order code=os28044 quantity=2]Link Text</A>
```

| *Parameters* | *Description* | *Default* |
|---|---|---|

| | | |
|---|---|---|
| code | This is the unique identifier for the item, typically the SKU in the products table | *none* |
| quantity | Quantity to order | 1 |

| ***Attributes*** | ***Default*** |
|---|---|
| interpolate (reparse) | *No* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Container tag | *No* |
| Has end tag | *No* (`[/order]` is a macro for `</A>`) |

**Tag expansion example:**

```
   [order os28044 2]Buy Framing Hammer[/order]
---
   <A HREF="http://localhost.localdomain/cgi-bin/tag72/ord/basket?\
   mv_session_id=6CZ2whqo&mv_pc=1&mv_action=refresh&\
   mv_order_item=os28044&mv_order_quantity=3">Buy Framing Hammer</A>
```

**ASP-like Perl call:**

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

## 2.50.2. 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.

## 2.50.3. How to Order an Item

Interchange can either use a form-based order or a link-based order to place an item in the shopping cart. The `order` tag creates a link-based order.

You can use the `area` tag with form variables if you need more control, for example, to change frames for the order:

```
   <A HREF="[area href=ord/basket
                  form="mv_order_item=os28044
                        mv_order_quantity=2
                        mv_action=refresh"]"
    TARGET=newframe> Order Framing Hammer</A>
```

To order with a form, you set the form variable `mv_order_item` to the item-code/SKU and use the `refresh` action:

```
   <FORM ACTION="[process-target]" METHOD=POST>
   <INPUT TYPE=hidden  NAME="mv_todo"        VALUE="refresh">
   <INPUT TYPE=hidden  NAME="mv_order_item"  VALUE="os28044">

   Order <INPUT NAME="mv_order_quantity" SIZE=3 VALUE=1> Framing Hammer

   <INPUT TYPE=submit VALUE="Order!">
   </FORM>
```

Groups of items may be batched:

```
<FORM ACTION="[process-target]" METHOD=POST>
<INPUT TYPE=hidden  NAME="mv_todo"        VALUE="refresh">

<INPUT TYPE=hidden  NAME="mv_order_item"  VALUE="TK112">
<INPUT NAME="mv_order_quantity" SIZE=3> Standard Toaster

<INPUT TYPE=hidden  NAME="mv_order_item"  VALUE="TK200">
<INPUT NAME="mv_order_quantity" SIZE=3> Super Toaster

<INPUT TYPE=submit VALUE="Order!">
</FORM>
```

Items that have a quantity of zero (or blank) will be skipped. Only items with a positive quantity will be placed in the basket.

Attributes like size or color may be specified at time of order. See the `accessories` tag for detail.

# 2.51. 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 </A>.

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>
```

## 2.51.1. Summary

```
[page href arg]
[page href=dir/page arg=page_arguments other_named_attributes]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| href | Path to Interchange page or action<br>*Special arguments*<br>♦ 'scan' treats **arg** as a search argument<br>♦ 'http://...' external link (requires **form** attribute) | process |
| arg | Interchange arguments to page or action | *none* |
| *base* | alias for **arg** | *none* |

| *Attributes* | *Default* |
|---|---|
| extra | *none* |
| form | *none* |
| search | *No* |

| | |
|---|---|
| secure | *No* |
| interpolate (reparse) | *No* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* ([/page] is a macro for </A>) |

**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);
```

Using arrayref for joined search (see also Attribute Arrays and Hashes)

```
my $searchref = [ "se=hammer/fi=products/sf=description",
                  "se=plutonium/fi=products/sf=description", ];

$Tag->page( { href   => 'scan',
              search => $searchref, } );
```

#### 2.51.1.1. See Also

area

## 2.51.2. 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.

#### 2.51.2.1. 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>
```

#### 2.51.2.2. 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"
      |]
```

See the Search and Form Variables appendix for a listing of the form variables along with two−letter abbreviations and descriptions.

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 `&#32;` 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).

### 2.51.2.3. 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.)

#### 2.51.2.4. 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.

#### 2.51.2.5. 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.

#### 2.51.2.6. Joined searches

You can also specify a joined search using an attribute array (see Attribute Arrays and Hashes):

```
[page href=scan
  search.0="se=fragrant
            fi=products
            sf=smell"
  search.1="se=purple
            sf=color"
  search.2="se=perennial
            sf=type"]
```

The search routine called by the page tag automatically adds the other relevant search specification elements, including the 'co=yes' to indicate a combined search (joined searches are described in the Interchange database documentation).

#### 2.51.2.7. [/page]

This is not an actual end tag, but simply a macro that expands to </A>. The following two lines are equivalent:

```
[page shirts]Our shirt collection[/page]
[page shirts]Our shirt collection</A>
```

Tip: In large pages, just use the </A> tag for a small performance improvement.

# 2.52. perl

Executes the perl code contained by the tag. The code will run under the restrictions of Perl's Safe module by default. The tag expands to the value returned by the enclosed code (i.e., printing to STDOUT or STDERR is useless).

See also Interchange Programming.

## 2.52.1. Summary

```
[perl tables] Code here [/perl]
[perl tables="db1 db2 ..." other_named_attributes] Code here [/perl]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| tables | Database tables to be made available to ASP Perl code | *none* |
| table | Alias for `tables` | *none* |

| *Attributes* | *Default* |
|---|---|
| failure | *none* |
| no_return | *No* |
| subs | *No* |
| arg="subs" | *Same as* `subs` |
| global | *No* |
| file | *none* |
| number_errors | *none* |
| eval_label | *none* |
| short_errors | *none* |
| trim_errors | *none* |
| interpolate | *No* |
| reparse | *Yes* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *Yes* |
| Has Subtags | *No* |
| Container tag | *Yes* |

**Tag expansion example:**

```
   [perl tables="products" failure="Perl code error <BR>"]
      my $result = "Looked up SKU $Values->{code}. It is a ";
      $result .= $Tag->data('products', 'description', $Values->{code} );
      return ("$result <br>\n");
   [/perl]
-------------------------------------------------------
   Looked up SKU os28044. It is a Framing Hammer <br>
```

**ASP−like Perl call:** (e.g., to use it like a runtime eval() within your code)

```
$Tag->perl(  { tables  => "products", },
             $code  );
```

or similarly with positional parameters,

```
$Tag->perl( $tables, $attribute_hash_reference );
```

#### 2.52.1.1. See Also

See also Interchange Programming, [calc], and [mvasp].

## 2.52.2. Description

This tag allows you to embed perl code within an Interchange page. The code will run under the restrictions of Perl's Safe module by default. Perl's '`warnings`' and '`strict`' pragmas are both turned off, and Safe will block you from turning them on, since it blocks Perl's '`use`' command. (This is not usually a problem, since you should probably use an alternative such as a usertag if your code is complex enough to need `strict`.)

The tag expands to the value returned by the enclosed code (i.e., printing to STDOUT or STDERR is useless).

```
[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>
```

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

| ***Object*** | ***Description*** |
|---|---|
| $CGI−>{key} | Hash reference to raw submitted values |
| $CGI_array−>{key} | Arrays of submitted values |
| $Carts−>{cartname} | Direct reference to shopping carts |
| $Config−>{key} | Direct reference to $Vend::Cfg |
| $DbSearch−>array(@args) | Do a DB search and get results |
| $Document−>header() | Writes header lines |
| $Document−>send() | Writes to output |
| $Document−>write() | Writes to page |
| $Scratch−>{key} | Direct reference to scratch area |

| | |
|---|---|
| $Session−>{key} | Direct reference to session area |
| $Tag−>tagname(@args) | Call a tag as a routine (UserTag too!) |
| $TextSearch−>array(@args) | Do a text search and get results |
| $Values−>{key} | Direct reference to user form values |
| $Variable−>{key} | Config variables (same as $Config−>{Variable}); |
| &HTML($html) | Same as $Document−>write($html); |
| &Log($msg) | Log to the error log |

For full descriptions of these objects, see Interchange Perl Objects.

#### 2.52.2.1. tables

This should ba a whitespace−separated list of database tables you want to make available within the Perl code.

If you wish to use database values in your Perl code, the tag must pre−open the table(s) you will be using. Here is an example using the products table:

```
[perl tables=products]
    my $cost = $Tag->data('products', 'our_cost', $Values->{code});
    $min_price = $cost * ( 1 + $min_margin );
    return ($min_price > $sale_price) ? $min_price : $sale_price;
[/perl]
```

If you do not do this, your code will fail with a runtime Safe error when it tries to look up 'our_cost' in the products database with the `data` tag.

Even if you properly specify the tables to pre−open, some database operations will still be restricted because `Safe` mode prohibits creation of new objects. For SQL, most operations can be performed if the `Safe::Hole` module is installed. Otherwise, you may have to set the `global=1` attribute to use data from SQL tables.

Interchange databases can always be accessed as long as they are pre−opened by using an item first.

Technical note:

Safe objects (including database handles) may persist within a page, and the `perl` tag does not necessarily destroy objects created earlier in the page. As a result, your code may work even though you did not set '`tables`' properly, only to break later when you change something elsewhere on the page.

For example, this will work because the first call to [`accessories` ...] opens the (default) products table:

```
[accessories code=os28044 attribute=size]

[perl]
  return $Tag->accessories( { attribute => 'size',
                              code      => 'os28085' } );
[/perl]
```

If you remove the first [accessories ...] tag, then the $Tag−>accessories call will fail with a Safe error unless you also set '`tables`=products' in the `perl` tag.

The moral of this story is to ensure that you pass all necessary tables in the `perl` tag.

#### 2.52.2.2. failure

If your code contains a compile or runtime error and fails to evaluate (i.e., eval($code) would set $@), the tag will return the value set for the `failure` attribute. The error will be logged as usual.

For example,

```
[perl failure="It Broke"]
    my $cost = $Tag->data('products', 'our_cost', $Values->{code});
    $min_price = $cost * ( 1 + $min_margin );
    return ($min_price > $sale_price) ? $min_price : $sale_price;
[/perl]
```

will return 'It Broke' because the $Tag->Data(...) call will fail under the Safe module (see tables above).

#### 2.52.2.3. no_return

If `no_return=1`, this attribute suppresses the return value of the perl code.

You can retrieve the return value from the session hash via `[data session mv_perl_result]` until it gets overwritten by another `perl` tag.

If `no_return` is set, the `perl` tag *will* return any output explicitly written with the &HTML or $Document->write() functions.

---

***Note:***

If `no_return` is *not* set, then the $Document->write() buffer is not returned (unless you use $Document->hot(1) or $Document->send(), in which case the contents of the write buffer will probably appear before anything else on the page). See Interchange Perl Objects for more detail.

---

Here is an example:

```
[perl tables=products no_return=1]
    my $cost = $Tag->data('products', 'our_cost', $Values->{code});
    $min_price = $cost * ( 1 + $min_margin );
    &HTML( ($min_price > $sale_price) ? $min_price : $sale_price );
    return ($min_price > $sale_price) ? 'too low' : 'ok';
[/perl]
```

This will put the same price on the page as our earlier example, but

`$Session->{mv_perl_result}` will be either 'too low' or 'ok'.

The [`mvasp`] tag is very similar to [perl no_return=1].

#### 2.52.2.4. subs

If you have set the `AllowGlobal` catalog directive, setting `subs=1` will enable you to call GlobalSub routines within the enclosed perl code. Note that this can compromise security.

#### 2.52.2.5. global

If you have set the `AllowGlobal` catalog directive, setting `global=1` will turn off Safe protection within the tag.

The code within the tag will then be able to do anything the user ID running Interchange can. This seriously compromises security, and you should know what you are doing before using it in a public site. It is especially dangerous if a single Interchange server is shared by multiple companies or user IDs.

Also, full '`use strict`' checking is turned on by default when in global mode. You can turn it off by using '`no strict;`' within your code. Note that any `strict` errors will go to the Interchange error logs, and the tag itself will fail silently within the page.

#### 2.52.2.6. file

This prepends the contents of the specified file or FileDatabase entry to the enclosed perl code (if any), then executes as usual.

For example,

```
[perl file="my_script.pl"][/perl]
```

would execute myscript.pl and expand to its return value.

Absolute filenames (or filenames containing '../') are prohibited by the `NoAbsolute` catalog directive.

If the filename is not absolute, Interchange first looks for a file in the current directory, then in the list set with the `TemplateDir` catalog directive. If it fails to find a file by that name, it then looks for an entry by that name in the database specified with the `FileDatabase` catalog directive.

#### 2.52.2.7. file

Add line numbers to the source code displayed in the error.log, amazingly useful if some of the perl is being generated elsewhere and interpolated.

#### 2.52.2.8. eval_label

Set to a string, will replace the (eval ###) in the error message with this label, handy to quickly track down bugs when you have more than one perl block in the page, especially if you are using short_errors.

#### 2.52.2.9. short_errors

If set to a true value, syntax errors and the like in perl tags will log just the error, not the whole source code of the block in question, handy when you have the code open in an editor anyway and don't want the error itself to get scrolled away when running 'tail −f error.log'.

#### 2.52.2.10. trim_errors

If set to a number, and the error produced includes a line number, then only that number of lines before and after the broken line itself will be displayed, instead of the whole block.

# 2.53. price

## 2.53.1. 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−like Perl call:**

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

 OR

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

Attribute aliases

```
            base ==> mv_ib
```

## 2.53.2. Description

Arguments:

```
        code       Product code/SKU
        base       Only search in product table *base*
        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 *q2*
5-9      Column *q5*
10-24    Column *q10*
25 up    Column *q25*
```

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:

```
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*.

# 2.54. process

This is a shortcut for the 'process' action, expanding to your catalog URL and session ID. It is analogous to the `area` tag, but is more limited. The following expansion is illuastrative:

```
   [process target=targetframe]
−−−
   http://www.here.com/cgi−bin/mycatalog/process.html?\
   id=6CZ2whqo" TARGET="targetframe
```

(the trailing backslash indicates continuation, i.e., the result should be only one line)

Note the mismatched quotes in the expansion. Your surrounding HTML should supply the containing quotes, like this:

```
   <A HREF="[process target=targetframe]">...
```

Alias: **process_target**

## 2.54.1. Summary

```
[process target secure]
[process target=targetframe secure=1 other_named_attributes]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| target | The target frame or window | *None* |
| secure | Boolean. If true (secure=1), the URL will link to your secure server. | *No* |

| *Attributes* | *Default* |
|---|---|
| interpolate (reparse) | *No* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Container tag | *No* |

**Tag expansion example:**

```
   [process targetframe 1]
−−−
   http://secure.here.com/cgi−bin/mycatalog/process.html?\
   id=6CZ2whqo" TARGET="targetframe
```

**ASP−like Perl call:**

```
$Tag->process(  { target => 'frametarget',
                  secure => 1, } );
```

or similarly with positional parameters,

```
$Tag->process($target, $secure, $attribute_hash_reference);
```

# 2.55. query

Passes SQL statements through to SQL databases, or allows SQL queries via Interchange's database abstraction into non−SQL databases and text files. The latter use may require Jochen Wiedmann's SQL Statement module (included with Bundle::Interchange from CPAN).

## 2.55.1. Summary

```
[query sql]
[query sql="SQL_query_text" other_named_attributes]
```

| ***Parameters*** | ***Description*** | ***Default*** |
|---|---|---|
| sql | The SQL statement.<br>• Passed directly through to an SQL database.<br>• For a non−SQL table, the tag interprets your SQL first. See Jochen Wiedmann's SQL Statement module for limitations and detail. | *none* |
| query | Alias for `sql` | *none* |

| ***Attributes*** | ***Default*** |
|---|---|
| table | products |
| base (alias for `table`) | products |
| type (row_count, html, list, textref) | *none: uses arrayref="" if no type* |
| arrayref | *arrayref="" if no* `type` *given* |
| hashref | *none* |
| more (type=list) | *No* |
| xx form var. abbrev. (type=list) | *see form variable* |
| _ (type=list) | sql |
| list_prefix (type=list) | list |
| random (type=list) | *No* |
| safe_data (type=list) | *No* |
| label (type=list) | current |
| form (type=list) | *none* |
| wantarray | *No* |
| interpolate | *No* |
| reparse | *Yes* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |

| | |
|---|---|
| Container tag | *Yes* |
| Has subtags | *Yes* |
| Nests | *No* |

**Tag usage example:**

This will list sku, description and price for ten products per page, followed by hyperlinks to the other pages of the list. Note that you may interpolate Interchange tags in the usual way if you double−quote the SQL statement.

```
[query sql="select sku, description, price from products where price < [value mv_arg]"
      type=list
      more=1
        ml=10]

   [on_match]Matched<br>[/on_match]
   [no_match]Not Found<br>[/no_match]

   [list]
     [sql−code] [sql−param description] [sql−price]
   [/list]

   [more_list]
     [more]
   [/more_list]
[/query]
```

**ASP−like Perl call:**

```
my $sql = "select * from products order by price";
my $result_array = $Tag->query( { sql => $sql,  },
                                $body );
my ($same_results, $col_name_hash, $col_name_array) =
                   $Tag->query( { sql => $sql,  },
                                $body );

my $result_hasharray = $Tag->query( { sql     => $sql,
                                      hashref => 'my_results',  },
                                    $body );
```

or similarly with positional parameters,

```
$Tag->query( $sql, $attribute_hash_reference, $body);
```

## 2.55.2. Description

The `query` tag allows you to make SQL queries. If you are using an SQL database table, the tag will pass your SQL statement directly to the database and return the result.

If your table is not in an SQL database (for example, GDBM, text, LDAP, and in−memory tables), Interchange will internally convert it to an Interchange search specification with Jochen Wiedmann's SQL Statement module (included with Bundle::Interchange from CPAN). This means that you can use simple SQL queries regardless of the underlying database implementation.

### 2.55.2.1. Subtags

For list queries (type=list), the following subtags are available:

| *Subtag* | *Usage* |
|---|---|
| on_match | `[on_match]`<br>`  do this if something matched`<br>`[/on_match]` |
| no_match | `[no_match]`<br>`  do this if nothing matched`<br>`[/no_match]` |
| list | `[list_prefix]`<br>`  do this for each matched item`<br>`[/list_prefix]`<br><br>The 'list' subtag defines a region where you can use any of the looping subtags that work in array−list context (see Looping tags and Sub−tags).<br><br>The default looping tag prefix will be 'sql'. Note however that you can override this by setting the prefix attribute in the enclosing query tag.<br><br>Similarly, the list_prefix attribute renames the [list] subtag itself to the value you set (see list_prefix below). |
| more_list | `[more_list]`<br>`  [more]`<br>`[/more_list]`<br><br>The 'more_list' and 'more' subtags are used when paginating the query results (see 'more' attribute). The [more] subtag will expand to a list of links to the other pages of the query results. |

See also the example at the end of the Summary section above.

### 2.55.2.2. Perl and ASP usage

If you are calling $Tag−>query within a perl tag (or whenever the code is secured by the Safe.pm module), you must be sure to set the tables attribute properly in the enclosing perl tag (see the perl tag documentation for detail).

The types that return text to a page (i.e., row_count, html, and textref) work as usual, returning an appropriate string. Note that you may also have access to the results as an array reference in $Vend::Interpolate::Tmp−>{"} for the life of the page.

If you do not set a type, the tag will return a reference to an array of array references, since the default with no type is arrayref="".

If you call $Tag−>query in scalar context and set arrayref or hashref, it will return your results as a reference to an array of either arrayrefs or hashrefs, respectively (i.e., the same data structures you would get from Perl's DBI.pm module with fetchall_arrayref).

In list context, the first returned element is the aforementioned reference to your results. The second element is a hash reference to your column names, and the third element is an an array reference to the list of column names.

The following examples should be illustrative:

```
[perl tables=products]
  my $sql = "select sku, price, description from products
             where price < 10 order by price";

  my $results = $Tag->query( { sql => $sql, } );
  my ( $same_results, $col_name_hashref, $col_name_arrayref)
      = $Tag->query( { sql => $sql, } );

  my $hash_results = $Tag->query( {     sql => $sql,
                                    hashref => 'my_results' } );

  # $Vend::Interpolate::Tmp->{my_results} == $hash_results
  # $Vend::Interpolate::Tmp->{''} == $results == $same_results

  return $Tag->uneval( $results );
[/perl]
```

***Technical Note:*** The $Tag->query() call works a bit differently in GlobalSubs and UserTags than within a [perl] tag. Specifically, in a GlobalSub or global UserTag, if you call query() in list context and want the three references (i.e., results, column hash and column array), then you need to set the 'wantarray=1' attribute in the query() call. See the wantarray attribute.

#### 2.55.2.3. sql

This is the text of your SQL statement. The standard Interchange quoting rules apply. For example, use double quotes (") if you want to interpolate Interchange tags within your SQL statement, backticks (`) to calculate a value, *etc.*

```
[query sql="select description, price from products
            where price < [value mv_arg]" ...]
   ...
[/query]
```

#### 2.55.2.4. table

The table attribute sets the database to use for the query. The default will typically be the database containing the '`products`' table (unless you have changed the first entry in $Vend::Cfg->{ProductFiles}).

#### 2.55.2.5. type

If you are not setting the '`arrayref`' or '`hashref`' attributes, then the type attribute defines the way the query will return its results. The `type` should be one of the following:

| ***Type*** | ***Returns*** |
|---|---|
| html | The html type returns the results in an html table. You will need to supply the enclosing <TABLE ...> and </TABLE> html tags. The following is an example of typical usage:<br><br>`<TABLE>`<br>`  [query sql="select * from products`<br>`              where price > 12`<br>`              order by price"`<br>`        type=html]`<br>`  [/query]` |

| | |
|---|---|
| | </TABLE> |
| list | This allows you to use subtags to control the query output and pagination. See the Subtags section above for detail. |
| row_count | This causes the tag to return the number of rows in the query result. |
| textref | This causes the tag to return a the query results as a serialized array of arrays that Perl can evaluate with its **eval()** function. Here is an illustrative example:<br><br>`my $rows = eval( $Tag->query( { sql  => "select * from products"`<br>`type => "textref" } )`<br>`);`<br><br>`my $r3_c0 = $rows->[3]->[0];` |

If you do not specify a type, the tag will create an arrayref as if you had set 'arrayref="'".

#### 2.55.2.6. arrayref and hashref

If you set '`arrayref`=*keyname*' or '`hashref`=*keyname*', the query will not ruturn results to the page. Instead, it will place the results of your query in the $Vend::Interpolate::Tmp hash. Using '`arrayref`=my_query' sets $Vend::Interpolate::Tmp->{my_query} to refer to an array of array references, while '`hashref`=my_query' creates an array of hash references.

Note that this is useful only if you intend to access the results within Perl code (for example, within a [`perl`] tag), since there is no direct output to the returned page.

The $Vend::Interpolate::Tmp hash persists only for the life of the template page being processed. If you need the query results array reference to outlive the page, you will have to save the reference somewhere more persistent such as the $Session hash:

```
$Session->{my_query} = $Vend::Interpolate::Tmp->{my_query};
```

Beware the impact on perfomance if you do this with large result sets.

Technical note -- the string returned by the '`textref`' `type` will **eval**() to the '`arrayref`' data structure.

#### 2.55.2.7. more

Requires '`type`=list'.

You must set `more`=1 to properly paginate your results from `list` queries (see '`type`=list' above. If you do not set `more`=1, then the links to later pages will merely redisplay the first page of your results.

#### 2.55.2.8. *form variable abbreviations*

Requires '`type`=list'.

See the Search and Form Variables appendix for a list of form variables. Note that you must use the two-letter abbreviation rather than the full form variable name.

A few deserve special mention:

| *Abbr* | *Name* | *Description* |
|---|---|---|
| ml | mv_matchlimit | Sets number of rows to return. If paginating (more=1), sets rows returned per page. |
| fm | mv_first_match | Start displaying search at specified match |
| sp | mv_search_page | Sets the page for search display |
| st | mv_searchtype | Forces a specific search type (text, glimpse, db or sql), overriding the default determined from your database implementation. |

**2.55.2.9.**

Requires 'type=list'.

Setting 'prefix=foo' overrides the default prefix of 'sql' for loop subtags within a list region (see Looping tags and Sub−tags).

See the list_prefix attribute below for an illustrative example.

### 2.55.2.10. list_prefix

Requires 'type=list'.

Setting 'list_prefix=bar' overrides the default region tagname of 'list'. The best way to show this is by example. Compare the following two examples of list queries, the first using the defaults and the second with explicitly set prefix and list_prefix.

```
   [query sql="select sku, description, price from products
               where price < 20"
         type=list
         more=1
           ml=10]

      [on_match]Matched<br>[/on_match]
      [no_match]Not Found<br>[/no_match]

      [list]
        [sql-code] [sql-param description] [sql-price]
      [/list]

      [more_list]
        [more]
      [/more_list]
   [/query]

------------------------------------------------------------

   [query  sql="select sku, description, price from products
                where price < 20"
          type=list
        prefix=foo
   list_prefix=bar
          more=1
            ml=10]

      [on_match]Matched<br>[/on_match]
      [no_match]Not Found<br>[/no_match]

      [bar]
```

```
      [foo-code] [foo-param description] [foo-price]
    [/bar]

    [more_list]
      [more]
    [/more_list]
  [/query]
```

#### 2.55.2.11. random

Requires '`type`=list'.

You can use the '`random`' attribute to randomly select a set of rows from the whole result set of your query. In other words, setting '`random`=*n*', where *n* > 0, causes the [`list`] region to loop over *n* randomly chosen rows rather than the full query result set.

The example below would display three randomly chosen products priced under 20.

```
  [query sql="select * from products
              where price < 20"
        type=list
     random=3]

    [list]
      [sql-code] [sql-param description] [sql-price]
    [/list]

  [/query]
```

#### 2.55.2.12. safe_data

Requires '`type`=list'.

Note -- you should not set this unless you need it and know what you are doing.

Setting '`safe_data`=1' allows the [sql-data] tag to return values containing the '[' character. See also Looping tags and Sub-tags.

Beware of reparsing issues.

#### 2.55.2.13. label

Requires '`type`=list'.

If you are setting up multiple simultaneously active search objects within a page, this allows you to distinguish them. The default label is 'current'. Most people will not need this.

#### 2.55.2.14. form

Requires '`type`=list'.

You can use this to pass one CGI form variable in the pagination links of a [more-list]. For example, '`form`="foo=bar"' to include '&foo=bar' in the URL of each of the pagination links.

Note that the variable will not be available in the initial result set since the query returns the first page directly (i.e., you did not follow a pagination link).

#### 2.55.2.15. wantarray

This is relevant only when calling $Tag−>query( ... ) within global Perl code such as a globalsub or global usertag where $MVSAFE::Safe is not defined. In these cases, setting '`wantarray`=1' allows the call to

```
$Tag−>query( { wantarray => 1, ... }, ... );
```

to return references as it would if called within an ordinary [`perl`] tag. Note that it does not force list context if you call $Tag−>query in scalar context.

Technical note −− the ordinary [`query` ...] ... [`/query`] usage forces scalar context on the query call and suppresses the return value for those `types` that would return references if $Tag−>query were called within a [`perl`] tag. The wantarray option is needed because global subs and usertags are also affected by this unless you set wantarray.

## 2.56. read_cookie

Returns the value of the named cookie. Returns nothing if the cookie does not exist.

### 2.56.1. Summary

```
[read_cookie name]
[read_cookie name=mycookie]
```

| *Attributes* | *Description* | *Default* |
|---|---|---|
| name | The name of the cookie whose value you want | *none* |

| *Attributes* | *Default* |
|---|---|
| interpolate (reparse) | *No* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *Yes* |
| Container tag | *No* |

**Usage example:**

```
   [read−cookie name=MV_SESSION_ID]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
   6CZ2whqo
```

**ASP−like Perl call:**

```
$Tag−>read_cookie(  { name => $name, }  );
```

or similarly with positional parameters,

```
$Tag−>read_cookie( $name );
```

## 2.56.2. Description

This tag expands to the value of the named cookie (or nothing if the cookie does not exist).

See the Netscape specification at http://www.netscape.com/newsref/std/cookie_spec.html if you need more cookie−specific detail.

### 2.56.2.1. name

This is the name of the cookie whose value you want to retrieve.

### 2.56.2.2. Parsing an HTTP_COOKIE string

If you pass this tag a second parameter within a Perl call, it will use your value as the HTTP_COOKIE string (ignoring the real one). This only applies if you pass the values positionally within a perl call since there is no name for the HTTP_COOKIE string input:

```
$Tag->read_cookie('MV_SESSION_ID', "MV_SESSION_ID=UnHyaDQj:127.0.0.1; ...");
```

# 2.57. restrict

Restrict tag execution in a region. If a restricted tag is encountered, it is simply output.

## 2.57.1. Summary

```
[restrict tag1 tag2]
[restrict policy=deny enable="page area value"]
```

| *Attributes* | *Description* | *Default* |
|---|---|---|
| policy | Whether to allow or deny by default. | **deny** |
| enable | Tags to enable when default policy is deny. | *none* |
| disable | Tags to disable. Overrides enable. | *none* |

| *Attributes* | *Default* |
|---|---|
| interpolate (reparse) | *No* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *Yes* |
| Container tag | *No* |

**Usage example:**

```
   [read-cookie name=MV_SESSION_ID]
-------------------------------------------------------
   6CZ2whqo
```

**ASP−like Perl call:**

```
    N/A. Cannot be called effectively.
```

## 2.57.2. Description

Restrict tag execution in a region. If a restricted tag is encountered, it is simply output. It can be used to allow certain tags in a user−editable region, while denying dangerous tags. Or it can be used to restrict all tag execution in a region.

#### 2.57.2.1. policy

Default is `deny`, which makes most sense. You then specifically enable certain ITL tags. If you set `allow` by default, you must be very careful that you really are disabling all of what you consider to be dangerous tags.

#### 2.57.2.2. enable

A space−separated or comma−separated list of tags to disable when the default policy is `deny`. Has no effect when the default policy is `allow`, and any tags passed in the disable parameter override the enable.

#### 2.57.2.3. disable

A space−separated or comma−separated list of tags to disable when the default policy is `allow`. If you have a list of tags that are enabled, perhaps stored in a scratch variable, you can disable some of those tags since this takes precedence over the enable.

# 2.58. row

## 2.58.1. 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−like Perl call:**

```
    $Tag->row(
        {
         width => VALUE,
        },
        BODY
    )

OR
```

```
$Tag->row($width, $BODY);
```

## 2.58.2. 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.

### 2.58.2.1. [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.
```

### 2.58.2.2. [/col]

Terminates the column field.

# 2.59. salestax

## 2.59.1. 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−like Perl call:**

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

 OR

    $Tag->salestax($name, $noformat);
```

Attribute aliases

```
            cart ==> name
```

## 2.59.2. 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.

# 2.60. scratch

## 2.60.1. 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−like Perl call:**

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

 OR

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

### 2.60.2. Description

Returns the contents of a scratch variable to the page. (A scratch variable is set with a [set] value [/set] container pair.)

## 2.61. scratchd

Deletes the named scratch variable and returns its value before the deletion. For example,

```
[scratchd varname_to_delete]
```

deletes the scratch variable *varname_to_delete*.

See also the `scratch` and `set` tags.

### 2.61.1. Summary

```
[scratchd P_PARAM]
[scratchd N_PARAM other_named_attributes]
```

| ***Parameters*** | ***Description*** | ***Default*** |
|---|---|---|
| name | Name of scratch variable to delete | *None* |

| ***Attributes*** | ***Default*** |
|---|---|
| interpolate (reparse) | *No* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *Yes* |
| Container tag | *No* |

**Tag expansion example:**

```
   [set myvar]This is myvar[/set]
   .
   .
   .
   [scratchd myvar]
---
   This is myvar
```

**ASP−like Perl call:**

```
    $Tag->scratchd($name, $attribute_hash_reference);
```

### 2.61.2. Description

Deletes the named scratch variable and returns its value before the deletion.

## 2.62. search_list

Formats results returned by a search. Must be enclosed within a `search region`. Has sub−tags (see `Looping tags and Sub−tags`).

# 2.63. search_region

## 2.63.1. 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−like Perl call:**

```
    $Tag->search_region(
        {
         arg => VALUE,
        },
        BODY
    )

 OR

    $Tag->search_region($arg, $ATTRHASH, $BODY);
```

Attribute aliases

```
            args ==> arg
            params ==> arg
            search ==> arg
```

## 2.63.2. Description

**NO Description**

# 2.64. selected

## 2.64.1. 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−like Perl call:**

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

OR

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

## 2.64.2. 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`.

# 2.65. set

## 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.

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

Invalidates cache: **YES**

Called Routine:

**ASP−like Perl call:**

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

OR

    $Tag->set($name, $BODY);
```

## 2.65.2. 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.

# 2.66. set_cookie

Sets browser cookie(s) with the specified attributes.

## 2.66.1. Summary

```
[set_cookie named_attributes]
```

Parameters must be named (no positional usage except in Perl call)

| *Attributes* | *Description* | *Default* |
|---|---|---|
| name | The name you give the cookie | *none* |
| value | The value (automatically html−escaped by Interchange) | *none* |
| expire | Expiration date as<br><br>"Wdy, DD−Mon−YYYY HH:MM:SS GMT" | *none* |
| domain | Overrides the domain(s) set in `CookieDomain` | *Domain(s), if any, defined in the* `CookieDomain` *directive* |
| path | legal URL paths for the cookie | *URL path(s) to your catalog, including aliases* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *Yes* |
| Container tag | *No* |

**Usage example:**

```
   [set−cookie name=mycookie
              value="the value"
             expire="Tue, 03−Apr−2001 17:00:00 GMT" ]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
   This tag returns no value in the page
```

**ASP−like Perl call:**

```
    $Tag−>set_cookie(  { name   => $name,
                         value  => $value,
                         expire => $expire,
                         domain => $domain,
                         path   => $path, }  );
```

or similarly with positional parameters,

```
    $Tag−>set_cookie( $name, $value, $expire, $domain, $path );
```

## 2.66.2. Description

This tag sets one or more browser cookies with your specified name, value, and expiration. (Interchange will set more than one cookie if needed to ensure that the cookie is visible from all `Catalog` URL path aliases and `CookieDomains`.)

See the Netscape specification at http://www.netscape.com/newsref/std/cookie_spec.html for more cookie−specific detail.

If you need access to the cookie from outside of your Interchange catalog, you can also set the domain and URL paths for which the cookie will be valid. If you need the cookie only within your catalog and the domains specified by the `CookieDomain` directive, you probably should not override the Interchange domain and path defaults.

#### 2.66.2.1. name

This is the name of the cookie. This is the key you will use when reading the cookie later.

#### 2.66.2.2. value

This is the value to store in the cookie.

#### 2.66.2.3. expire

Persistent cookies (that outlive a browser session) require an expiration date. The date must be a string of the form:

"Wdy, DD−Mon−YYYY HH:MM:SS GMT"

and the timezone must be GMT.

If you do not supply a date, the cookie will disappear when the user closes the browser.

#### 2.66.2.4. domain

The value you set will override the Interchange default domain(s). You might set this if you need access to the cookie from outside the Interchange catalog, but it is usually better to set the `CookieDomain` directive in your catalog.

The default is to use your catalog's domain or all `CookieDomain` values.

#### 2.66.2.5. path

The value you set will override the Interchange default URL path(s).

The default is to set a cookie with a path for each catalog alias (see the `Catalog` directive). This ensures that the cookie will be visible regardless of how the end user returns to your catalog.

# 2.67. seti

## 2.67.1. 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−like Perl call:**

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

OR

    $Tag->seti($name, $BODY);
```

## 2.67.2. Description

Equivalent to the [`set`] tag, except that it interpolates by default.

# 2.68. setlocale

Sets locale and/or currency for the current page. Can be made persistent for the user with the 'persist' option. Resets default locale if called without arguments. See also Setting the Locale in the template documentation.

## 2.68.1. Summary

**Parameters: locale currency**

- locale
  - The locale to use.
  - Default: [scratch mv_locale] (see also 'persist' attribute)
- currency
  - The currency format to use.
  - Default: [scratch mv_currency] (see also 'persist' attribute)

Positional parameters in same order.

**Named Attrbutes**

- persist
  - Setting '`persist=1`' also sets the scratch variables, **mv_locale** and **mv_currency** to specified locale and currency. This makes the locale settings persistent for the user's session. Otherwise (`persist=0`), the `setlocale` tag affects the remainder of the current page

only.

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

**ASP−like Perl call:**

```
    $Tag->setlocale(
        {
         locale => VALUE,
         currency => VALUE,
        }
    )

OR

    $Tag->setlocale($locale, $currency, $ATTRHASH);
```

## 2.68.2. Description

Immediately sets the locale to `locale`, and will cause it to persist in future user pages if the `persist` is set to a non−zero, non−blank value. If the `currency` attribute is set, the pricing and currency−specific locale keys and Interchange configuration directives are modified to that locale. If there are no arguments, it sets it back to the user's default locale as defined in the scratch variables `mv_locale` and `mv_currency`.

This allows:

```
    Dollar Pricing:

    [setlocale en_US]
    [item-list]
    [item-code]: [item-price]<BR>
    [/item-list]

    Franc Pricing:

    [setlocale fr_FR]
    [item-list]
    [item-code]: [item-price]<BR>
    [/item-list]

    [comment] Return to the user's default locale [/comment]
    [setlocale]
```

# 2.69. shipping

Returns the cost of shipping the items in the cart via the shipping mode set with the **mode** parameter. See also the Shipping section of the Database documentation.

## 2.69.1. Summary

Parameters: **mode**

- mode
  - Aliases: **name**, **modes**
  - Whitespace, null or comma delimited list of modes for which to calculate shipping cost. See also mv_shipmode.
  - Default: [value mv_handling] if **handling=1** or [value mv_shipmode] or 'default'
- table
  - Alias: **tables**
  - Whitespace−delimited list of tables containing shipping data required for perl or query calculations (*e.g.*, in the 'PERL' field of your shipping database −− see Shipping). You must specify the tables to get past the Perl 'Safe.pm' proctection. For example, you will get 'Safe' errors if you refer to an SQL table without specifying it here.
  - Default: None
- cart
  - Alias: **carts**
  - Comma−delimited list of names of carts to calculate shipping cost for.
  - Default: current cart
- convert
  - Applies the conversion (if any) set with the `PriceDivide` catalog configuration directive.
  - Default: True
- noformat
  - Returns result as a number rather than a string formatted for the current locale's currency.
  - Default: True
- handling
  - Boolean−− use [value mv_handling] rather than [value mv_shipping] as first default for **mode**. Note that this attribute matters only if you do not specify the **mode** in the tag.
  - Note that this is set by the [`handling` tag (which calls the `shipping` tag internally). You should probably use the handling tag rather than setting this directly.
  - Default: False
- reset_modes
  - Clears list of modes in $Vend::Cfg−>{Shipping_line}
  - Default: False
- add
  - Adds the argument to **add** as data for a shipping.asc file (in $Vend::Cfg−>{ScratchDir}/) and sets it.
- file
  - Filename to read shipping from (default is usual shipping database, e.g., shipping.asc)
- label
  - By default, returns HTML <OPTION ...> widget for shipping mode(s), including description and cost. You can override the widget with the **format** attribute. Note that **label** overrides **noformat**.
  - Here is an example from the foundation checkout.html page:

```
[shipping
  label=1
  mode=|[data table=country key='[default country US]' col=shipmodes]|
]
```

- format
  - Format for results with **label** attribute.
  - Default: '<OPTION VALUE="%M"%S>%D (%F)'
  - For example,
    [shipping mode="FLAT"
    label=1
    format="My Format Desc %D Price %F"]
- default
  - Resets shipping mode to default of [value mv_shipmode]
- hide
  - Suppresses output
- reset_message
  - Boolean. Blanks the session's current shipping message (i.e., $Session−>{ship_message}).

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

**ASP−like Perl call:**

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

OR

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

Attribute aliases

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

## 2.69.2. Description

This tag calculates the shipping cost for items in the current cart via the specified shipping mode (usually set in the mv_shipmode variable). See the Shipping section of the Database documentation for detail.

### 2.69.2.1. Rounding

Note that the tag rounds the calculated shipping cost to a locale−specific number of fractional digits (e.g., to the nearest penny, or 2 digits after the decimal point in the USA).

# 2.70. shipping_desc

Returns the shipping description for the specified shipping **mode**. See the Shipping section of the Database

documentation. See also shipping.asc database for shipping modes.

Alias: **shipping_description**

The two tags below are identical in operation:

```
[shipping_desc mode]
[shipping_description mode]
```

## 2.70.1. Summary

```
[shipping_desc mode]
[shipping_desc mode=shipping_mode]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| mode | Shipping mode. This is a key into the shipping.asc database. See Shipping documentation. | `mv_shipmode`, or 'default' if `mv_shipmode` not set |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | Yes, but only if no **mode** given |
| Container tag | *No* |

**Tag expansion example:**

```
[shipping_desc 1DM]
---
UPS Next Day Air Early AM
```

**ASP−like Perl call:**

```
$Tag->shipping_desc( $mode );
```

# 2.71. soap

## 2.71.1. 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−like Perl call:**

```
$Tag->soap(
```

```
        {
         call => VALUE,
         uri => VALUE,
         proxy => VALUE,
        }
    )

OR

    $Tag->soap($call, $uri, $proxy, $ATTRHASH);
```

### 2.71.2. Description

**NO Description**

## 2.72. strip

Strips leading and trailing whitespace from the contained body text.

### 2.72.1. Summary

```
[strip]
   Body text to strip
[/strip]
```

No parameters.

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Container tag | *Yes* |
| Has Subtags | *No* |

**ASP−like Perl call:**

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

or even better, just do it directly like this

```
$BODY =~ s/^\s+//;
$BODY =~ s/\s+$//;
```

## 2.73. subtotal

### 2.73.1. 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−like Perl call:**

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

OR

    $Tag->subtotal($name, $noformat);
```

Attribute aliases

```
            cart ==> name
```

## 2.73.2. 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.

# 2.74. tag

## 2.74.1. 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−like Perl call:**

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

OR

    $Tag->tag($op, $arg, $ATTRHASH, $BODY);
```

Attribute aliases

```
            description ==> arg
```

## 2.74.2. Description

Performs any of a number of operations, based on the presence of `arg`. The arguments that may be given are:

### 2.74.2.1. 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`.

### 2.74.2.2. 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).

#### 2.74.2.3. 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`.

#### 2.74.2.4. 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.

#### 2.74.2.5. mime boundary

Returns a MIME message boundary with unique string keyed on session ID, page count, and time.

#### 2.74.2.6. mime header

Returns a MIME message header with the proper boundary for that session ID, page count, and time.

#### 2.74.2.7. mime id

Returns a MIME message id with the proper boundary for that session ID, page count, and time.

#### 2.74.2.8. 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]
```

#### 2.74.2.9. 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]
```

#### 2.74.2.10. 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]
```

# 2.75. time

## 2.75.1. 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−like Perl call:**

```
$Tag->time(
    {
     locale => VALUE,
    },
    BODY
)
```

OR

```
$Tag->time($locale, $ATTRHASH, $BODY);
```

## 2.75.2. 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:

#### 2.75.2.1. 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.

#### 2.75.2.2. 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.

#### 2.75.2.3. 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
```

#### 2.75.2.4. 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.

#### 2.75.2.5. zerofix

Strips leading zeroes from numbers.

# 2.76. timed_build

## 2.76.1. 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−like Perl call:**

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

OR

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

## 2.76.2. 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:

### 2.76.2.1. auto

Turns on automatic region processing. The text of the timed−build region is processed to determine a unique checksum or digest (using MD5), and that file name is checked in the directory tmp/auto−timed (assuming `ScratchDir` is set to `tmp`). If no number of minutes is supplied, `60` is assumed.

This is designed to automatically build regions of commonly used areas without having to manage the file name yourself.

Implies `login=1`, but will still abort if no session ID cookie has been sent. Use `force=1` to ignore cookie status.

### 2.76.2.2. file

Name of the file to save the results in. Relative to catalog root. The directory must exist.

### 2.76.2.3. if

Allows you to to only display the cached region when the `if` paremeter 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.

#### 2.76.2.4. 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.

#### 2.76.2.5. 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.

# 2.77. tmp

## 2.77.1. 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−like Perl call:**

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

```
        BODY
    )

OR

    $Tag->tmp($name, $BODY);
```

## 2.77.2. 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`.

# 2.78. total_cost

## 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−like Perl call:**

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

OR

    $Tag->total_cost($name, $noformat);
```

Attribute aliases

```
            cart ==> name
```

## 2.78.2. Description

Expands into the total cost of all the items in the current shopping cart, including sales tax (if any).

# 2.79. tree

## 2.79.1. 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−like Perl call:**

```
    $Tag->tree(
        {
         table => VALUE,
         master => VALUE,
         subordinate => VALUE,
         start => VALUE,
        },
        BODY
    )

 OR

    $Tag->tree($table, $master, $subordinate, $start, $ATTRHASH, $BODY);
```

Attribute aliases

```
            sub ==> subordinate
```

## 2.79.2. 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.

#### 2.79.2.1. mv_level

Level of the item. If it is in the first level, it is 0. Sublevels are infinite (except for performance).

#### 2.79.2.2. 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.

#### 2.79.2.3. mv_children

If in autodetect mode, set to the number of children this branch has. If a leaf, set to 0.

#### 2.79.2.4. 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
        master=parent_fld
        subordinate=child_fld
        autodetect=1
                spacing=4
        full=1]
    <tr>
```

```
<td>
[if-item-param mv_level]
        [item-calc]
                return ' ' 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 paremeters:

#### 2.79.2.5. table

Database table which contains the tree. Must be a valid Interchange table identifier.

#### 2.79.2.6. master

The column which is used to determine the parent of the item.

#### 2.79.2.7. 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 `master`.

#### 2.79.2.8. start_item

The first item to be followed, i.e. the `master` value of all the top-level items.

#### 2.79.2.9. 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.

#### 2.79.2.10. 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.

#### 2.79.2.11. stop

An optional `stop` field which, when the value is true, can stop the following of the branch.

#### 2.79.2.12. continue

An optional `continue` field which, when the value is true, can force the branch to be followed.

#### 2.79.2.13. sort

The column which should be used for ordering the items −− determines the order in which they will be displayed under the current parent.

#### 2.79.2.14. 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
```

#### 2.79.2.15. memo

Indicates that the collapse/explode/toggle features are to be used, and names a `Scratch` variable where the values should be stored.

#### 2.79.2.16. 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.

#### 2.79.2.17. 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.

#### 2.79.2.18. 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.

#### 2.79.2.19. 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.

#### 2.79.2.20. 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.

#### 2.79.2.21. 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.

## 2.80. try

Allows you to trap errors. Interchange processes the body text of the [`try`][`/try`] block and returns it normally if it does not generate an error. If it does generate an error, interchange executes the [`catch`][`/catch`] block.

See also '`catch`'.

### 2.80.1. Summary

```
[try label=my_label other_named_attributes]
    Body text to return if no error
[/try]
.
.
.
[catch my_label]
    Body text to return if try block caused an error
[/catch]
```

| *Parameters* | *Description* | *Default* |
|---|---|---|
| label | The label shared by the paired `try` and `catch` blocks | 'default' |
| ***Attributes*** | ***Description*** | ***Default*** |

| | | |
|---|---|---|
| status | Returns 0 (failed) or 1 (succeeded) instead of page output | *none* |
| hide | Suppresses page output | *No* |
| clean | Suppress `try` block output if it has an error | *No* |
| interpolate | See Interpolating Parameters | *No* |
| reparse | See Interpolating Parameters | *Yes* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Container tag | *Yes* |

**Tag expansion example:**

```
   [set divisor]0[/set]
   [try label=div]
      [calc] 1 / [scratch divisor] [/calc]
   [/try]
   [catch div]Division error[/catch]
---
   Division Error
```

**ASP−like Perl call:**

```
    $Tag->try(    { label  => I<'try_catch_label'>,
                    status => 1, },
                  $try_body  );

    $Tag->catch(  { label  => I<'try_catch_label'>, },
                  $catch_body )
```

or similarly with positional parameters,

```
    $Tag->try($label, $attribute_hash_reference, $try_body);
    $Tag->catch($label, $attribute_hash_reference, $catch_body);
```

#### 2.80.1.1. See Also

`catch`

## 2.80.2. Description

Allows you to trap errors. Interchange processes the body text of the [`try`][`/try`] block and returns it normally if it does not generate an error. If it does generate an error, interchange executes the [`catch`][`/catch`] block. The catch block executes where it is on the page (*i.e.*, it does not replace the output of the try block).

Note that the `catch` block must occur after the [try] block in the document.

#### 2.80.2.1. label

The try and catch blocks are matched by this label.

Technical note:

The try tag will also place a result in the $Session object. For example, the following returns the 'Illegal division by zero...' error message if it occurs:

```
[try label=divide][calc] 1 / [scratch divisor] [/calc][/try]

[catch divide]
   [calc]$Session->{try}{divide}[/calc]
[/catch]
```

The $Session->{try}{divide} object will be set to the empty string ('') if there was no error, or it will contain the error message if there was an error.

The [perl] and [calc] tags also set $Session->{try}->{*active_label*} on errors.

#### 2.80.2.2. status

Suppresses `try` block output and returns 1 if no error or 0 if an error occurred instead. Executes the `catch` block as usual in case of an error.

#### 2.80.2.3. hide

Suppresses `try` block output (regardless of success or failure). Executes the `catch` block as usual in case of an error.

#### 2.80.2.4. clean

Setting '`clean=1`' will cause the `try` block to suppress its output only if it has an error. Otherwise (`clean=0` or not set), the `try` block will return whatever partial output it has completed before the error. The `catch` block will work as usual.

## 2.81. update

Forces an update of the specified interchange function. Function may be one of the following:

- **cart** (updates current or named cart)
- **process** (updates order or search)
- **values** (updates user-entered fields)
- **data** (updates database, using current **mv_** CGI form variables)

### 2.81.1. Summary

Parameters: **function**

- function
  - ♦ cart
    - ◊ Upates current or named cart (see `name` attribute)
  - ♦ process
    - ◊ Updates an order or a search page
  - ♦ values
    - ◊ Updates user-entered fields
  - ♦ data

◊ Updates database, using current **mv_** CGI form variables, for example:
· **mv_data_table** Table to update
· **mv_data_key** Key into table
· **mv_data_fields** Fields to update (space or null delimited)
· **mv_data_function** One of the following:
- delete
- update
- insert
- delete

· etc.

- name
  - ♦ Cart name to update (if 'function=cart')
  - ♦ Default: current cart

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

**ASP−like Perl call:**

```
    $Tag−>update(
        {
         function => VALUE,
        }
    )

OR

    $Tag−>update($function, $ATTRHASH);
```

## 2.81.2. Description

Forces an update of the specified interchange function. Function may be one of the following:

- **cart** (updates current or named cart)
- **process** (updates order or search)
- **values** (updates user−entered fields)
- **data** (updates database, using current **mv_** CGI form variables)

# 2.82. userdb

## 2.82.1. 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−like Perl call:**

```
    $Tag−>userdb(
        {
         function => VALUE,
        }
    )

 OR

    $Tag−>userdb($function, $ATTRHASH);
```

Attribute aliases

```
            name ==> nickname
            table ==> db
```

## 2.82.2. 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):

```
database     Sets user database table (userdb)
show         Show the return value of certain functions
             or the error message, if any (0)
force_lower  Force possibly upper-case database fields
             to lower case session variable names (0)
billing      Set the billing fields (see Accounts)
shipping     Set the shipping fields (see Address Book)
preferences  Set the preferences fields (see Preferences)
bill_field   Set field name for accounts (accounts)
addr_field   Set field name for address book (address_book)
pref_field   Set field name for preferences (preferences)
cart_field   Set field name for cart storage (carts)
pass_field   Set field name for password (password)
time_field   Set field for storing last login time (time)
expire_field Set field for expiration date (expire_date)
acl          Set field for simple access control storage (acl)
file_acl     Set field for file access control storage (file_acl)
db_acl       Set field for database access control storage (db_acl)
```

# 2.83. value

Returns the the current value of the named form input field. HTML-escapes Interchange tags in the result for security.

Can also set a new value within the current page.

## 2.83.1. Summary

```
[value name]
[value name=form_var_name other_named_attributes]
```

| ***Parameters*** | ***Description*** | ***Default*** |
|---|---|---|
| name | This is the name of the form variable whose value you want. | *None* |

| ***Attributes*** | ***Default*** |
|---|---|
| set | *none* |
| hide | *No* |
| filter | *none* |
| keep (with filter) | *No* |
| scratch | *No* |
| default | *none* |
| interpolate (reparse) | *No* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *Yes* |

**Tag expansion example:**

Assuming form variable 'foo' = 'bar',

```
   [value foo]
---
   bar
```

**ASP−like Perl call:**

```
    $Tag->value(  { name => var_name, }  );

    # or if you simply want the value,
    $::Values->{var_name};
```

or similarly with positional parameters,

```
    $Tag->value($name$, $attribute_hash_reference);
```

## 2.83.2. Description

HTML examples:

```
   <PARAM MV="value name">
   <INPUT TYPE="text" NAME="name" VALUE="[value name]">
```

Expands into the current value of the named customer/form input field. 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.

### 2.83.2.1. name

This is the name of the form variable whose value you want.

### 2.83.2.2. set

You can change a value with '`set`=*new_value*'. The tag will return the value you set unless you also set the hide=1 attribute.

Use this to "uncheck" a checkbox or set other form variable values to defaults. If you simply want a place to store your own data, use the `set` and `scratch` tags instead.

Note that this is only available in new−style tags, for safety reasons.

### 2.83.2.3. hide

Setting `hide`=1 suppresses the tag's return value, which can be useful with the `set`set attribute.

### 2.83.2.4. filter

See the `filter` tag for a list of filters.

Setting '`filter`="*filter*"' modifies the named value with the specified filter.

#### 2.83.2.5. keep (with filter)

Set `keep`=1 if you want the tag to return a filtered result but do not want the filter to modify the form value itself in the $::Values hash.

#### 2.83.2.6. scratch

Setting '`scratch`=1' places a copy of the value in the $::Scratch hash. An illustrative example:

```
foo is [value name=foo scratch=1] in the Values hash
foo is now also [scratch foo] in the Scratch hash
```

#### 2.83.2.7. default

This sets a return value in case the named value is missing or otherwise false. The following will expand to "Using default":

```
[value name=myname set=0 hide=1]
[value myname default="Using default"]
```

# 2.84. value_extended

## 2.84.1. 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−like Perl call:**

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

OR

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

## 2.84.2. 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`.

### 2.84.2.1. 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.

### 2.84.2.2. joiner

The character or string that will join the elements of the array. Will accept string literals such as "\n" or "\r".

### 2.84.2.3. 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.

### 2.84.2.4. 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.

### 2.84.2.5. 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.

#### 2.84.2.6. 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.

#### 2.84.2.7. 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.

#### 2.84.2.8. 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.

#### 2.84.2.9. 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]
```

As is the case with [perl] tag, user tags run under the Perl Safe.pm module with warnings disabled. Unlike [perl] tags, however, user tags use Perl's 'strict' pragma.

The properties for UserTag 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 [`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.noinsert=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 <A HREF=...> 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 intepreted.

## 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
```

<BR> 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
```

```
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 Usertags

The distribution includes a number of prebuilt usertags in the `usertag` directory in the Interchange software directory. Some of these are used by the foundation catalog or its administrative interface.

## 4.1. bar_button

Displays content based on current page. Could be used for building e.g. menu bars.

### 4.1.1. Summary

```
[bar-button page current]...[selected]...[/selected][/bar-button]
[bar-button page=page current=current-page]...[selected]...[/selected][/bar-button]
```

***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*** |
|---|---|---|
| page | Name of page for which this bar-button is defined | *none* |
| current | Name of the current page | Current page: MV_PAGE |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | [/bar-button] |

**Tag expansion example:**

To build a simple '3-button' menu bar one could put the following on each of the pages. The results of this code for page2 are shown below.

```
        <table><tr>
        [bar-button page=page1]
        <TD><A HREF="[area page1]">PAGE-1</A></TD>
        [selected]
        <TD bgcolor="red"><A HREF="[area page1]"><B>PAGE-1-selected</B></A></TD>
        [/selected]
        [/bar-button]
        [bar-button page=page2]
        <TD><A HREF="[area page2]">PAGE-2</A></TD>
        [selected]
        <TD bgcolor="red"><A HREF="[area page2]"><B>PAGE-2-selected</B></A></TD>
        [/selected]
        [/bar-button]
        [bar-button page=page3]
        <TD><A HREF="[area page3]">PAGE-3</A></TD>
        [selected]
        <TD bgcolor="red"><A HREF="[area page3]"><B>PAGE-3-selected</B></A></TD>
        [/selected]
        [/bar-button]
        </tr></table>
--------------------------------------------------------
        PAGE-1    PAGE-2-selected    PAGE-3
```

**ASP−like Perl call:**

```
$Tag->button_bar(  { page => $page,
                current  => $current,
                    body => $body}  );
```

or similarly,

```
$Tag->area($page, $current, $body);
```

#### 4.1.1.1. See Also

bar_link routine

### 4.1.2. Description

Displays content based on current page. The content between the [selected][/selected] tags will be displayed only if the name of the curent page matches the name that was passed to the page parameter (page=page−name). The default content will be displayed when there is no match.

#### 4.1.2.1. page

The name of the page for which this definition of the bar−button is defined.

#### 4.1.2.2. current

The name of the current page. Defaults to current page MV_PAGE.

## 4.2. button

## 4.3. convert_date

This tag converts a given date format into another format.

### 4.3.1. Summary

```
[convert_date day* other_named_attributes>[/convert_date]
[convert_date day=n* other_named_attributes[/convert_date]
```

***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* |
|---|---|---|
| days | The number of days from or before today | *none* |

| *Attributes* | *Default* |
|---|---|
| format | '%d−%b−%Y %I:%M%p' or '%d−%b−%Y' |
| fmt − Alias for format | *none* |
| raw | *none* |

| zerofix | *none* |
|---|---|
| ***Other_Charactreristics*** | |
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | [/convert_date] |

**Tag expansion example:**

```
  a. [convert-date][/convert-date]
  b. [convert-date 1][/convert-date]
  c. [convert-date -1][/convert-date]
  d. [convert-date]2001-5-1[/convert-date]
  e. [convert-date]2001-05-01[/convert-date]
  f. [convert-date]20010515[/convert-date]
  g. [convert-date raw=1]2001-05-18[/convert-date]
  h. [convert-date fmt="%d-%m-%Y"]2001-05-18[/convert-date]
  i. [convert-date]200 1 -  --051 =9[/convert-date]
  j. [convert-date]2001 -  --05 -20 11 1 5[/convert-date]
  k. [convert-date raw=1]2001-05-21 11:15[/convert-date]
-------------------------------------------------------
  a. 18-May-2001 03:15AM (todays day and time)
  b. 19-May-2001 03:15AM (today + 1 day)
  c. 17-May-2001 03:15AM (today - 1 day)
  d. 01-May-2001
  e. 01-May-2001
  f. 15-May-2001
  g. 20010518
  h. 18-05-2001
  i. 19-May-2001
  j. 20-May-2001 11:15AM
  k. 200105211115
```

**ASP-like Perl call:**

```
    $Tag->convert_date( { day => 1 } );

    $Tag->convert_date( { body => "2001-05-19 15:35",
                          format => "%d-%m-%Y %H:%M", } );
```

or similarly with positional parameters,

```
    $Tag->convert_date( 1 );
```

## 4.3.2. Description

This tag converts almost any given date format into another, possibly user defined, format.

### 4.3.2.1. days

Number of days from or before today's date and time. Will only be used if nothing is supplied between the tags.

#### 4.3.2.2. format

POSIX time format string of your choice. See Unix strftime(3) manpage for complete details.

#### 4.3.2.3. raw

If this option is set to true, will display given date in raw format, e.g. yyyymmdd or yyyymmddHHMM.

#### 4.3.2.4. zerofix

Strips leading zeroes from numbers.

## 4.4. db_date

This tag returns the time of last access of the database source file.

### 4.4.1. Summary

```
[db_date table* format*]
```

***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*** |
|---|---|---|
| table | Table name. | products |
| format | POSIX time format string | %A %d %b %Y |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* |

**Tag expansion example:**

```
   [db-date]
   [db-date cat]
   [db-date table=cat format="%d %b %Y"]
-----------------------------------------------------
   Wednesday 02 May 2001 (products.txt)
   Wednesday 03 May 2001 (cat.txt)
   03 May 2001 (cat.txt)
```

**ASP-like Perl call:**

```
    $Tag->db_date( { table => cat,
                     format => "%d %b %Y", } );
```

or similarly with positional parameters,

```
    $Tag->db_date( "cat", "%d %b %Y" );
```

## 4.4.2. Description

This tag returns the time of last access of the database source file.

#### 4.4.2.1. table

Table name. Defaults to products if not specified.

#### 4.4.2.2. format

POSIX time format string. See Unix strftime(3) manpage for details. Defaults to '%A %d %b %Y' when not specified.

# 4.5. delete_cart

This tag deletes a cart from the userdb.

## 4.5.1. Summary

```
[delete_cart nickname]
[delete_cart nickname="cart-name"]
```

***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*** |
|---|---|---|
| nickname | Must be an existing nickname | *none* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* |

**Tag expansion example:**

```
   [delete_cart mycart]
   [delete_cart nickname="mycart"]
-----------------------------------------------------
```

**ASP-like Perl call:**

```
$Tag->delete_cart( { nickname => "mycart", } );
```

or similarly with positional parameters,

```
$Tag->delete_cart( "mycart" );
```

#### 4.5.1.1. See Also

userdb, load_cart, save_cart and pages templates/components/saved_carts_list_small, pages/saved_carts.html for more examples.

### 4.5.2. Description

Deletes a cart with name nickname from the user database. Basically the same as [userdb function=delete_cart nickname=mcart].

#### 4.5.2.1. nickname

Nickname of cart to be deleted.

## 4.6. email

This tag takes a recipient address and a body text and uses the SendmailProgram with −t option to send email.

### 4.6.1. Summary

```
[email to subject* reply* from* extra*]Your message[/email]
[email to=to_address subject=subject reply=reply_address
    from=from_address extra=extra_headers]Your message[/email]
```

***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*** |
|---|---|---|
| to | Email address of recipient | *none* |
| subject | Subject line | String: <no subject> |
| reply | Email address to be used for the reply−to header | *none* |
| from | Senders email address | First address in MailOrderTo configuration variable |
| extra | Additional headers to be included | *none* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | [/email] |

**Tag expansion example:**

```
[email
       to="foo@bar.com"
  subject="Greetings"
     from="bar@foo.com"
]
Hello World
[/email]
```

---

**ASP−like Perl call:**

```
$Tag->email(  { to => $to,
```

```
                from => $from,
             subject => $subject,
               reply => $reply,
               extra => $extra,
                body => $body }  );
```

or similarly,

```
$Tag->email($to, $subject, $reply, $from, $extra, $body);
```

#### 4.6.1.1. See Also

email_raw and etc/mail_receipt, pages/process_return.html, pages/stock−alert−added.html for examples.

### 4.6.2. Description

Will send the content between the [email][/email] tags as an email to the recipient (to) using the SendmailProgram with −t option.

#### 4.6.2.1. extra

Extra headers to be included. Example: Errors−To: errors@yourdomain.com

#### 4.6.2.2. from

Email address identifying the sender of the message. Will use the first email address of the MailOrderTo configuration variable if it is not supplied.

#### 4.6.2.3. reply

Email address to be used for the Reply−to header. No Reply−to header will be present if this parameter is omitted.

#### 4.6.2.4. subject

Short text describing the content of the message. The Subject line of an email message. The string <no subject> will be substituted if this parameter is omitted.

#### 4.6.2.5. to

Valid email address(es) of the recipient(s). This parameter is required.

## 4.7. email_raw

This tag takes a raw email message, **including headers**, and uses the SendmailProgram with −t option.

### 4.7.1. Summary

```
[email_raw]Your message incuding headers[/email_raw]
```

| *Other_Charactreristics* | |
|---|---|

| | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | [/email_raw] |

**Tag expansion example:**

```
   [email_raw]
From: foo@bar.com
To: bar@foo.com
Subject: baz

The text of the message.
   [/email_raw]
----------------------------------------------------
```

The headers must be at the beginning of the line, and the header must have a valid To: or it will not be delivered.

**ASP−like Perl call:**

```
    $Tag->email_raw(  { body => $body }  );
```

or similarly,

```
    $Tag->email_raw($body);
```

#### 4.7.1.1. See Also

email

### 4.7.2. Description

Will send the content between the [email_raw][/email_raw] tags as a raw email message to the recipient specified in the supplied headers using the SendmailProgram with −t option.

## 4.8. fcounter

## 4.9. fedex_query

## 4.10. formel

## 4.11. get−url

Fetch a URL and return the contents.

### 4.11.1. Summary

```
    [get-url url]
    [get-url url="valid-url" strip=1*]
```

***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*** |
|---|---|---|
| url | Must be a valid URL. Meaning, you have to supply the protocol.<br><br>Example<br><br>♦ http://demo.akopia.com/<br>♦ ftp://ftp.akopia.com/ | *none* |

| ***Attributes*** | ***Default*** |
|---|---|
| strip | *none* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* |

**Tag expansion example:**

```
   [get-url http://demo.akopia.com/]
   [get-url url="http://demo.akopia.com/" strip=1]
-----------------------------------------------------
```

**ASP-like Perl call:**

```
    $Tag->get_url( { url => "http://demo.akopia.com/", } );

    $Tag->get_url( { url => "http://demo.akopia.com/",
                     strip => 1, } );
```

or similarly with positional parameters,

```
    $Tag->get_url( "http://demo.akopia.com/" );
```

## 4.11.2. Description

Uses the LWP libraries (LWP::Simple) to fetch a URL and returns the contents.

### 4.11.2.1. strip

If the strip option is set, strips everything up to <body> and everything after </body>.

### 4.11.2.2. url

Must be a valid URL (including protocol).

# 4.12. load_cart

This tag loads a cart by name from the userdb.

## 4.12.1. Summary

```
[load_cart nickname]
[load_cart nickname="cart-name"]
```

***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* |
|---|---|---|
| nickname | Must be an existing nickname.<br><br>Nickname is constructed from:<br><br>♦ a name part<br>♦ time modified (time the cart was saved by save_cart tag)<br>♦ type (c for cart, r for recurring) | *none* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* |

**Tag expansion example:**

```
   [load_cart mycart:990102732:c]
   [load_cart nickname="mycart:990102732:c"]
-----------------------------------------------------
```

**ASP-like Perl call:**

```
$Tag->load_cart( { nickname => "mycart:990102732:c", } );
```

or similarly with positional parameters,

```
$Tag->load_cart( "mycart:990102732:c" );
```

### 4.12.1.1. See Also

userdb, delete_cart, save_cart and pages templates/components/saved_carts_list_small, pages/saved_carts.html for more examples.

### 4.12.2. Description

Loads a cart with name nickname from the user database. It will be merged with the current cart. Basically the same as [userdb function=get_cart nickname=cartname merge=1].

#### 4.12.2.1. nickname

Nickname of cart to be loaded. See above.

## 4.13. loc

## 4.14. rand

## 4.15. reconfig

## 4.16. reconfig_time

## 4.17. reconfig_wait

## 4.18. save_cart

This tag saves the current cart or recurring order in the userdb under a given name.

### 4.18.1. Summary

```
[save_cart nickname recurring]
[save_cart nickname="cart-name" recurring=1]
```

***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*** |
|---|---|---|
| nickname | Label for the cart. | *none* |
| recurring | Set to true if recurring. Set to false, or ommit if cart. | *none* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* |

**Tag expansion example:**

```
   [save_cart mycart]
   [save_cart nickname=mycart recurring=1]
```

---

**ASP-like Perl call:**

```
$Tag->save_cart( { nickname => mycart,
                   recurring => 1, } );
```

or similarly with positional parameters,

```
$Tag->save_cart( "mycart", "1" );
```

#### 4.18.1.1. See Also

userdb, delete_cart, load_cart and pages templates/components/saved_carts_list_small, pages/saved_carts.html for more examples.

## 4.18.2. Description

Saves the current cart with name nickname in the user database. Basically the same as [userdb function=set_cart nickname=cartname]

#### 4.18.2.1. nickname

Nickname for the current cart to be saved. You can use same nickname for different carts. An index will be added if there are more carts with the same nickname.

#### 4.18.2.2. recurring

Set to true if recurring. Set to false, or simply ommit it, if it is a cart.

# 4.19. summary

This tag calculates column totals.

## 4.19.1. Summary

```
[summary amount]
[summary amount=n.nn other_named_attributes]
```

***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*** |
|---|---|---|
| amount | Numerical value to be added to previous total | *none* |

| ***Attributes*** | ***Default*** |
|---|---|
| currency | *none* |
| format | *none* |
| hide | *none*, no hiding |
| name | ONLY0000, internal use only |
| reset | *none* |
| total | *none* |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | *No* |

**Tag expansion example:**

```
   [loop list="10 20 30.5"]
   [summary amount="[loop-code]" hide=1]
   [/loop]
   [summary total=1 format="%.3f"]
   [summary total=1 currency=1]
-------------------------------------------------------
   60.500
   $60.50
```

**ASP-like Perl call:**

```
    $Tag->summary( { amount => 10.5,
                     hide => 1, } );

    $Tag->summary( { amount => 25,
                     name => mytotal,
                 currency => 1, } );
```

or similarly with positional parameters,

```
    $Tag->summary( 10.5, $attribute_hash_reference );
```

#### 4.19.1.1. See Also

templates/components/cart, pages/ord/checkout.html for more examples.

## 4.19.2. Description

The summary tag provides you with an easy way to calculate and display totals. The display of the amounts is fully customizable. You can hide diplay, or you can show the amounts with the proper currency formatting according to the locale, or you can define your own formatting. Any number of summaries can be kept on a page.

#### 4.19.2.1. currency

The amount or total will be diplayed according to the currency formatting of the current locale if this attribute is set to true (non blank or zero).

#### 4.19.2.2. format

You can choose any formatting of the amount you like. Just set the format attribute to the desired formatting string (%s, %.2f etc.). When both, currency and format attributes are set, the format attribute will take precedence. So it doesn't make much sense to set them both at the same time.

#### 4.19.2.3. hide

Will suppress the display of amount when set to true.

#### 4.19.2.4. name

You can calculate as many totals as you like on the same page. Just supply a different label for each summary.

#### 4.19.2.5. reset

Will erase the total(s) if set to true. Be careful tough. It will reset ALL totals when you have no name attribute supplied. If you have provided a label for the name atribute then it will only reset the total for that particular label. All others won't be touched.

#### 4.19.2.6. total

Will show the total instead of the amount if set to true.

# 4.20. table_organize

Takes an unorganized set of table cells and organizes them into rows based on the number of columns.

## 4.20.1. Summary

```
[table-organize cols* other_named_attributes]
         [loop ....] <td> [loop-tags] </td> [/loop]
    [/table-organize]

[table-organize cols=n* other_named_attributes]
         [loop ....] <td> [loop-tags] </td> [/loop]
    [/table-organize]
```

---

***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* |
|---|---|---|
| cols | Number of columns. | 2 |
| columns | Alias for cols. | 2 |

| *Attributes* | *Default* |
|---|---|
| caption | *none* |
| columnize | *none* |
| embed | *none* |
| filler |   |
| limit | *none* |
| pretty | *none* |
| rows | *none* |
| table | *none* |

| | |
|---|---|
| td | *none* |
| tr | *none* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | [/table−organize] |

**Tag expansion example:**

This example produces a table that (1) alternates rows with background colors "#EEEEEE" and "#FFFFFF", and (2) aligns the columns right, center, left:

```
        [table−organize
            cols=3
            pretty=1
            tr.0='bgcolor="#EEEEEE"'
            tr.1='bgcolor="#FFFFFF"'
            td.0='align=right'
            td.1='align=center'
            td.2='align=left'
            ]
            [loop list="1 2 3 1a 2a 3a 1b"] <td> [loop−code] </td> [/loop]
        [/table−organize]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
        <tr bgcolor="#EEEEEE">
                <td align=right>1</td>
                <td align=center>2</td>
                <td align=left>3</td>
        </tr>
        <tr bgcolor="#FFFFFF">
                <td align=right>1a</td>
                <td align=center>2a</td>
                <td align=left>3a</td>
        </tr>
        <tr bgcolor="#EEEEEE">
                <td align=right>1b</td>
                <td align=center> </td>
                <td align=left> </td>
        </tr>
```

If the attribute columnize=1 is present, the result will look like:

```
            <tr bgcolor="#EEEEEE">
                    <td align=right>1</td>
                    <td align=center>1a</td>
                    <td align=left>1b</td>
            </tr>
            <tr bgcolor="#FFFFFF">
                    <td align=right>2</td>
                    <td align=center>2a</td>
                    <td align=left> </td>
            </tr>
            <tr bgcolor="#EEEEEE">
                    <td align=right>3</td>
                    <td align=center>3a</td>
                    <td align=left> </td>
            </tr>
```

See the source for more ideas on how to extend this tag.

**ASP−like Perl call:**

```
$Tag−>table_organize( { cols => 3,
                        pretty => 1, }, $BODY );
```

or similarly with positional parameters:

```
$Tag−>table_organize( $cols, $attribute_hash_reference, $BODY );
```

#### 4.20.1.1. See Also

pages/flypage.html, pages/quantity.html, templates/components/best_horizontal, templates/components/cart, templates/components/cross_horizontal, templates/components/random, templates/components/random_vertical, templates/components/upsell

## 4.20.2. Description

Takes an unorganized set of table cells and organizes them into rows based on the number of columns; it will also break them into separate tables.

If the number of cells are not on an even modulus of the number of columns, then "filler" cells are pushed on.

#### 4.20.2.1. cols (or columns)

Number of columns. This argument defaults to 2 if not present.

#### 4.20.2.2. rows

Optional number of rows. Implies "table" parameter.

#### 4.20.2.3. table

If present, will cause a surrounding <TABLE> </TABLE> pair with the attributes specified in this option.

#### 4.20.2.4. caption

Table <CAPTION> container text, if any. Can be an array.

#### 4.20.2.5. td

Attributes for table cells. Can be an array.

#### 4.20.2.6. tr

Attributes for table rows. Can be an array.

#### 4.20.2.7. columnize

Will display cells in (newspaper) column order, i.e. rotated.

#### 4.20.2.8. pretty

Adds newline and tab characters to provide some reasonable indenting.

#### 4.20.2.9. filler

Contents to place in empty cells put on as filler. Defaults to " ".

#### 4.20.2.10. limit

Maximum number of cells to use. Truncates extra cells silently.

#### 4.20.2.11. embed

If you want to embed other tables inside, make sure they are called with lower case <td> elements, then set the embed tag and make the cells you wish to organize be <TD> elements. To switch that sense, and make the upper−case or mixed case be the ignored cells, set the embed parameter to "lc".

```
[table−organize embed=lc]
            <td>
                    <TABLE>
                            <TR>
                            <TD> something
                            </TD>
                            </TR>
                    </TABLE>
            </td>
[/table−organize]
```

or

```
[table−organize embed=uc]
            <TD>
                    <table>
                            <tr>
                            <td> something
                            </td>
                            </tr>
                    </table>
            </TD>
[/table−organize]
```

The "tr", "td", and "caption" attributes can be specified with indexes; if they are, then they will alternate according to the modulus.

The "td" option array size should probably always equal the number of columns; if it is bigger, then trailing elements are ignored. If it is smaller, no attribute is used.

# 4.21. title_bar

Creates a quick title bar.

## 4.21.1. Summary

```
[title-bar width size color]My title[/title-bar]
[title-bar width=600 size=5 color="#ff0000"]My title[/title-bar]
```

***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* |
|---|---|---|
| color | Background color the bar.<br>Defaults to<br>♦ variable HEADERBG or<br>♦ #444444 | HEADERBG or #444444 |
| size | Font size | 6 |
| width | Width of the title bar | 500 |

| *Other_Charactreristics* | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | [/title-bar] |

**Tag expansion example:**

```
   [title-bar 600 5 red]My title[/title-bar]
   [title-bar width=600 size=5 color="#ff0000"]My title[/title-bar]
-----------------------------------------------------
```

**ASP-like Perl call:**

```
    $Tag->title_bar( { body => "My Title", } );

    $Tag->title_bar( { width => 400,
                       color => "#0000ff",
                        body => "My title", } );
```

or similarly with positional parameters,

```
    $Tag->title_bar( 600, 5, "red", "My title" );
```

## 4.21.2. Description

Quickly adds a title bar to your pages without having to type the html each time. Background color, width of the bar and size of the text can be customized by setting the appropriate parameter. The text color defaults to variable HEADERTEXT or when its not present to white.

#### 4.21.2.1. color

Sets the background color of the bar. You can set the color as 'red', '#ff0000', or 'bgcolor="#ff0000"'.

#### 4.21.2.2. size

Determines the size of the text. Parameter shoud be set to a value accepted by the HTML <font> tag size attribute.

#### 4.21.2.3. width

Sets the width of the bar.

# 4.22. ups_query

# 4.23. usertrack

# 4.24. var

# 4.25. xml_generator

This is a quick and dirty tag that generates XML tags based upon one of two types of data (delimited and session).

## 4.25.1. Summary

```
[xml-generator type* other_named_attributes][/xml-generator]
[xml-generator type=value* other_named_attributes][/xml-generator]
[xml-generator type=value* other_named_attributes][][/xml-generator]

*Optional
```

***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* |
|---|---|---|
| type | Data type. Delimited or session | delimited |

| *Attributes* | *Default* |
|---|---|
| attributes | *none* |
| dbdump | *none* |
| delimiter | \t |

| | |
|---|---|
| field_names |   |
| separator | \n |
| toplevel_tag | 'table' for delimited type and 'session' for other type |
| record_tag | record |
| field_tag | field |
| key_name | *none* |
| spacer | [\s,]+ |
| no_second | *none* |
| skip_empty | *none* |

| ***Other_Charactreristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| Has end tag | [/xml−generator] |

**Tag expansion example:**

```
[xml−generator
    type=delimited
    attributes="date"
    date="[tag time]%d−%b−%Y[/tag]"
    toplevel_tag=products]code  description     price
[query list=1 sql="select sku, description, price from products" prefix=xml][xml−code]  [xml−param descr
[/query][/xml−generator]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
<products date="18−May−2001">
        <record key="os28113">
                <code>os28113</code>
                <description>The Claw Hand Rake</description>
                <price>14.99</price>
        </record>
        <record key="os28006">
                <code>os28006</code>
                <description>Painters Brush Set</description>
                <price>29.99</price>
        </record>
        ...
</products>
```

**ASP−like Perl call:**

```
    $Tag−>xml_generator( {type => delimited,
                      toplevel_tag => apex,  }, $BODY );
```

or similarly with positional parameters,

```
    $Tag−>xml_generator( $type, $attribute_hash_reference, $BODY );
```

## 4.25.2. Description

#### 4.25.2.1. type

delimited

Accepts a delimited and separated (default is TAB delimiter and newline separator) list of records such as that generated by an '[item−list]', '[sql]', or '[loop search=""]' ITL tag.

session

When the type is not delimited, it can contain any hash reference into the Interchange session. Examples are:

```
values       The form values
scratch      Scratch values
errors       Error values
other        Any other Session key, for example "source" for
             [data session source]
```

If the value is a hash, then it will be sent as an XML record with the top level equal to "session", and a second_level tag equal to the hash name, and keys as separate XML container tags. If the parameter "that is equal to the type" is given, only those fields will be shown. Otherwise the entire hash will be shown. For example, this tag:

```
[xml−generator type="values" values="fname lname"][/xml−generator]
```

will generate:

```
<session>
        <values>
                <fname>First</fname>
                <lname>Last</lname>
        </values>
</session>
```

if it is a scalar, then only the second level will be done:

```
[xml−generator type="cybercash_id"][/xml−generator]
```

will do the equivalent of:

```
<session>
        <cybercash_id>[data session cybercash_id]</cybercash_id>
</session>
```

So bringing it all together, the following:

```
[xml−generator type="values scratch source"
               values="fname lname"
               scratch="downloads"][/xml−generator]
```

will generate:

```
<session>
        <values>
                <fname>First</fname>
                <lname>Last</lname>
        </values>
        <scratch>
                <downloads>0</downloads>
        </scratch>
```

```
                <source>Partner1</source>
        </session>
```

#### 4.25.2.2. no_second

Prevents the second−level tags from being generated. Extending the last example in the "session" type above, this

```
        [xml−generator  type="values scratch source"
                        no_second=1
                        values="fname lname"
                        scratch="downloads"][/xml−generator]
will generate:

        <session>
                <fname>First</fname>
                <lname>Last</lname>
                <downloads>0</downloads>
                <source>Partner1</source>
        </session>
```

#### 4.25.2.3. attributes

The attributes (if any) to pass on to the top level tag. For instance,

```
        [xml−generator
              attributes="date"
              date="[tag time]%d−%b−%Y[/tag]"
              toplevel_tag=order
        ][/xml−generator]

will generate a toplevel tag pair of:

        <order date="18−Mar−2001">
        </order>
```

#### 4.25.2.4. dbdump

Will dump all tables in the catalog when this attribute is set true. Used attributes are "toplevel_tag", "record_tag", "field_tag", and "skip_empty" or default values (resp. 'table', 'record', 'field').

```
Output format:
    <database name="catalogname">
        <toplevel_tag name="tablename1">
            <record_tag key="value of first field−1">
                <field_tag name="fieldname1">fieldvalue1</field_tag>
                <field_tag name="fieldname2">fieldvalue2</field_tag>
                        </record_tag>
            <record_tag key="value of first field−2">
                <field_tag name="fieldname1">fieldvalue1</field_tag>
                <field_tag name="fieldname2">fieldvalue2</field_tag>
                        </record_tag>
            </toplevel_tag>
        <toplevel_tag name="tablename2">
            <record_tag key="value of first field−1">
                <field_tag name="fieldname1">fieldvalue1</field_tag>
                <field_tag name="fieldname2">fieldvalue2</field_tag>
                        </record_tag>
```

```
        </toplevel_tag>
</database>
```

**Important note:** All tables are read into memory. So be warned, this could be a real memory hog.

Ton Verhagen's proposal:

1. Add option to select tables. E.g. dump_tables="products cat area" and/or
2. Add option to select an output file. E.g. dump_file="tabledump.XML". Send output to file line by line.

#### 4.25.2.5. delimiter

Character used as delimiter of fields in delimited data type. Defaults to a tab character.

#### 4.25.2.6. field_names

Space or comma−delimited list of field names to be used for delimited data type. Should be in the same order as in the data list provided (between the tags).

Another way of providing the field names would be:

```
[xml−generator .....]fieldname−1    fieldname−2     fieldname−3
    [field value list
     delimited by option delimiter and
     separated by option separator][/xml−generator]
```

---

***Note:*** Field name list **must** be tab delimited.

---

Ton Verhagen's humble opinion: This should change in future versions! Use option delimiter instead.

#### 4.25.2.7. separator

Character used as line separator in list between [xml−separator][xml−separator] tags and in output 'session' data type. Defaults to a newline, "\n".

#### 4.25.2.8. toplevel_tag

The toplevel tag name to use. Defaults to "table" for the 'dbdump mode' and delimited type, and "session" for the other.

#### 4.25.2.9. record_tag

Defines the tag name for the record tag. Defaults to 'record'. Used for 'dbdump mode' and delimited type.

#### 4.25.2.10. field_tag

Defines the tag name for the field tag. Defaults to 'field'. Only used in 'dbdump mode'.

#### 4.25.2.11. key_name

Only used in delimited data type. Defines fieldname to determine key value in "record_tag".

```
<record_tag key="value of field with name defined by key_name"> ....
```

#### 4.25.2.12. spacer

Character used as delimiter in type parameter definition and corresponding attributes. Defaults to '[\s,]+' (one or more whitespace or comma).

```
[xml-generator type="values|scratch"
                   values="value1|value2"
                              scratch="scratch1|scratch2"
                              spacer="|"
                              ][/xml-generator]
```

#### 4.25.2.13. skip_empty

Only used in dbdump mode (dbdump=1). Will skip empty fields if this attribute is set true.

# A. Tag Entry Format

## 4.26. dummy

Alias: **pedagogy**, **dummy_alias**

***Note:*** [**dummy** ...], [**pedagogy** ...] and [**dummy−alias** ...] are equivalent. If the tag has an endtag, you must not mix aliases between the tag and its endtag (i.e., [dummy] ... [/pedagogy] would not work).

A short description of the tag goes here. This example (**dummy**) is not an actual tag.

### A..1. Summary

```
[dummy first second][/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<br>*Special arguments*<br>♦ '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.<br>♦ '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 | *none* |
| *alias1* | 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 | *none* |
| still−more | *none* (requires **more**) |
| other | *none* |
| interpolate (body) | *Yes* |
| reparse | *No* |

| ***Other_Characteristics*** | |
|---|---|
| Invalidates cache | *No* |
| Macro | *No* |
| 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 ActedAy OnAy 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->dummy(  { first  => 'goober',
                    second => 'foobar',
                    more   => $snafu, },  $body_text  );
```

or similarly,

```
    $Tag->dummy($first, $second, $attribute_hash_reference, $body_text);
```

## 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

## B.1. Standard Parsing

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 between the HTML comment delimiters and the tag, 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. Hence, the *prefix*−tags are not handled recursively.
   - Some tags are interpreted in the lib/Vend/Parse.pm:start routine. You cannot call them with the '$Tag−>*tagname*' syntax. They are:
     - ◊ The [goto] tag. Note also that the goto tag handles the [label] tag.
     - ◊ The [bounce] tag.

8. Image paths substitution on the HTML output occurs.

## B.2. 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.

## B.3. Nonstandard parsing of Subtags

Subtags are parsed within the containing array−list or hash−list context created by the containing tag (see Looping tags and Sub−tags).

- All subtags at an earlier precedence level are treated before any in the next level.
- Within the same level, tags are proccessed in the order the appear on the page.
- Any standard tags are processed during 'interpolate' (before) or 'reparse' (after) phases of processing the containing tag.

---

***Technical note***

Processing within a hash− or array−list is actually done as a series of global regular expression substitutions on the page. Each substitution replaces one tag with the output of the subroutine(s) associated with it.

---

In array−list context, subtags are processed in the following order:

1. Check for *prefix*_**line** and prepare for it if present (does not process *prefix*−**line** at this time)
2. *prefix*−**sub** definitions processed
3. **if**−*prefix*−*etc.* nesting resolved
4. *prefix*−**alternate** processed
5. **if**−*prefix*−**param** processed
6. **if**−*prefix*−**pos** processed
7. *prefix*−**pos** processed
8. **if**−*prefix*−**field** processed
9. *prefix*−**line** processed
10. *prefix*−**increment** processed
11. *prefix*−**accessories** processed
12. *prefix*−**options** processed
13. *prefix*−**code** processed
14. *prefix*−**description** processed
15. *prefix*−**field** processed
16. *prefix*−**price** processed
17. *prefix*−**change** processed
18. *prefix*−**calc** processed
19. *prefix*−**exec** processed
20. *prefix*−**filter** processed
21. *prefix*−**last** processed
22. *prefix*−**next** processed
23. User's previous HTML widget SELECTED settings restored
24. Reparse standard tags in output of above (if reparse enabled for the containing tag)

In hash−list context, subtags are processed in the following order:

1. *prefix*−**sub** definitions processed
2. **if**−*prefix*−*etc.* nesting resolved
3. *prefix*−**alternate** processed
4. *prefix*−**line** processed

5. **if**−*prefix*−**param** processed
6. **if**−*prefix*−**field** processed
7. **if**−*prefix*−**modifier** processed (**if**−*prefix*−**param** and **if**−*prefix*−**modifier** are functionally identical except for parse order)
8. *prefix*−**increment** processed
9. *prefix*−**accessories** processed
10. *prefix*−**options** processed
11. *prefix*−**sku** processed
12. *prefix*−**code** processed
13. *prefix*−**quantity** processed
14. *prefix*−**modifier** processed
15. *prefix*−**param** processed
16. *prefix*−**quantity−name** processed
17. *prefix*−**modifier−name** processed
18. *prefix*−**subtotal** processed
19. *prefix*−**discount−subtotal** processed
20. *prefix*−**code** processed again differently (operating on new instances of *prefix*−**code** in output of above?)
21. *prefix*−**field** processed
22. *prefix*−**description** processed
23. *prefix*−**price** processed
24. *prefix*−**discount−price** processed
25. *prefix*−**difference** processed
26. *prefix*−**discount** processed
27. *prefix*−**change** processed
28. *prefix*−**tag** processed (*** CHECK THIS TAG NAME ***)
29. *prefix*−**calc** processed
30. *prefix*−**exec** processed
31. *prefix*−**filter** processed
32. *prefix*−**last** processed
33. *prefix*−**next** processed
34. User's previous HTML widget SELECTED settings restored
35. Reparse standard tags in output of above (if reparse enabled for the containing tag)

# C. Search and Form Variables

## C.1. Variable Names

### C.1..1. other

| *Name* | *scan* | *Type* | *Description* |
|---|---|---|---|
| mv_all_chars | ac | S | Turns on punctuation matching |
| mv_arg[0−9]+ | | A | Parameters for mv_subroutine (mv_arg0,mv_arg1,...) |
| mv_base_directory | bd | S | Sets base directory for search file names |
| mv_begin_string | bs | S | Pattern must match beginning of field |
| mv_case | cs | S | Turns on case sensitivity |
| mv_cartname | | O | Sets the shopping cart name |
| mv_cache_params | | S | Determines caching of searches |
| mv_change_frame | | A | Any form, changes frame target of form output |
| mv_check | | A | Any form, sets multiple user variables after update |
| mv_checkout | | O | Sets the checkout page |
| mv_click | | A | Any form, sets multiple form variables before update |
| mv_click | | XA | Default mv_click routine, click is mv_click_arg |
| mv_click *name* | | XA | Routine for a click *name*, sends click as arg |
| mv_click_arg | | XA | Argument name in scratch space |
| mv_coordinate | co | S | Enables field/spec matching coordination |
| mv_column_op | op | S | Operation for coordinated search |
| mv_credit_card* | | O | Discussed in order security (some are read−only) |
| mv_delay_page | dp | S | Delay search until after inital page display |
| mv_dict_end | de | S | Upper bound for binary search |
| mv_dict_fold | df | S | Non−case sensitive binary search |
| mv_dict_limit | di | S | Sets upper bound based on character position |
| mv_dict_look | dl | S | Search specification for binary search |
| mv_dict_order | do | S | Sets dictionary order mode |
| mv_doit | | A | Sets default action |
| mv_email | | O | Reply−to address for orders |
| mv_exact_match | em | S | Sets word−matching mode |
| mv_failpage | fp | O,S | Sets page to display on failed order check/search |
| mv_field_file | ff | S | Sets file to find field names for Glimpse |
| mv_field_names | fn | S | Sets field names for search, starting at 1 |
| mv_first_match | fm | S | Start displaying search at specified match |
| mv_head_skip | hs | S | Sets skipping of header line(s) in index |
| mv_index_delim | id | S | Delimiter for search fields (TAB default) |

| | | | |
|---|---|---|---|
| mv_matchlimit | ml | S | Sets match page size |
| mv_max_matches | mm | S | Sets maximum match return (only for Glimpse) |
| mv_min_string | ms | S | Sets minimum search spec size |
| mv_negate | ne | S | Records NOT matching will be found |
| mv_nextpage | np | A | Sets next page user will go to |
| mv_numeric | nu | S | Comparison numeric in coordinated search |
| mv_order_group | | O | Allows grouping of master item/sub item |
| mv_order_item | | O | Causes the order of an item |
| mv_order_number | | O | Order number of the last order (read−only) |
| mv_order_quantity | | O | Sets the quantity of an ordered item |
| mv_order_profile | | O | Selects the order check profile |
| mv_order_receipt | | O | Sets the receipt displayed |
| mv_order_report | | O | Sets the order report sent |
| mv_order_subject | | O | Sets the subject line of order email |
| mv_orsearch | os | S | Selects AND/OR of search words |
| mv_profile | mp | S | Selects search profile |
| mv_range_alpha | rg | S | Sets alphanumeric range searching |
| mv_range_look | rl | S | Sets the field to do a range check on |
| mv_range_max | rx | S | Upper bound of range check |
| mv_range_min | rm | S | Lower bound of range check |
| mv_record_delim | dr | S | Search index record delimiter |
| mv_return_all | ra | S | Return all lines found (subject to range search) |
| mv_return_delim | rd | S | Return record delimiter |
| mv_return_fields | rf | S | Fields to return on a search |
| mv_return_file_name | rn | S | Set return of file name for searches |
| mv_return_spec | rs | S | Return the search string as the only result |
| mv_save_session | | C | Set to non−zero to prevent expiration of user session |
| mv_search_field | sf | S | Sets the fields to be searched |
| mv_search_file | fi | S | Sets the file(s) to be searched |
| mv_search_line_return | lr | S | Each line is a return code (loop search) |
| mv_search_match_count | | S | Returns the number of matches found (read−only) |
| mv_search_page | sp | S | Sets the page for search display |
| mv_searchspec | se | S | Search specification |
| mv_searchtype | st | S | Sets search type (text, glimpse, db or sql) |
| mv_separate_items | | O | Sets separate order lines (one per item ordered) |
| mv_session_id | id | A | Suggests user session id (overridden by cookie) |
| mv_shipmode | | O | Sets shipping mode for custom shipping |
| mv_sort_field | tf | S | Field(s) to sort on |

| | | | |
|---|---|---|---|
| mv_sort_option | to | S | Options for sort |
| mv_spelling_errors | er | S | Number of spelling errors for Glimpse |
| mv_substring_match | su | S | Turns off word−matching mode |
| mv_successpage | | O | Page to display on successful order check |
| mv_todo | | A | Common to all forms, sets form action |
| mv_todo.map | | A | Contains form imagemap |
| mv_todo.checkout.x | | O | Causes checkout action on click of image |
| mv_todo.return.x | | O | Causes return action on click of image |
| mv_todo.submit.x | | O | Causes submit action on click of image |
| mv_todo.x | | A | Set by form imagemap |
| mv_todo.y | | A | Set by form imagemap |
| mv_unique | un | S | Return unique search results only |
| mv_value | va | S | Sets value on one−click search (va=var=value) |

## C.2. Abbreviations

The two−letter abbreviations are mapped with these letters:

| ***Abbr*** | ***Long name*** |
|---|---|
| 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 |
| 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 |