Template Guide

Table of Contents

Table of Contents

1. Introduction

Interchange is designed to build its pages based on templates from a database. This document describes how to build templates using the Interchange Tag Language (ITL) and explains the different options you can use in a template.

1.1. Overview

The search builder can be used to generate very complex reports on the database or to help in the construction of ITL templates. Select a "Base table" that will be the foundation for the report. Specify the maximum number of rows to be returned at one time, and whether to show only unique entries.

The "Search filter" narrows down the list of rows returned by matching table columns based on various criteria. Up to three separate conditions can be specified. The returned rows must match all criteria.

Finally, select any sorting options desired for displaying the results and narrow down the list of columns returned if desired. Clicking "Run" will run the search immediately and display the results. "Generate definition" will display an ITL tag that can be placed in a template and that will return the results when executed.

To build complex order forms and reports, Interchange has a complete tag language with over 80 different functions called Interchange Tag Language (ITL). It allows access to and control over any of an unlimited number of database tables, multiple shopping carts, user name/address information, discount, tax and shipping information, search of files and databases, and much more.

There is some limited conditional capability with the [if] tag, but when doing complex operations, use of embedded Perl/ASP should be strongly considered. Most of the tests use Perl code, but Interchange uses the Safe.pm module with its default restrictions to help ensure that improper code will not crash the server or modify the wrong data.

Perl can also be embedded within the page and, if given the proper permission by the system administrator, call upon resources from other computers and networks.

2. About Variable Replacement

Variable substitution is a simple and often used feature of Interchange templates. It allows you to set a variable to a particular value in the `catalog.cfg` directory. Then, by placing that variable name on a page, you invoke that value to be used. Before anything else is done on a template, all variable tokens are replaced by variable values. There are three types of variable tokens:

`__VARIABLENAME__` is replaced by the catalog variable called VARIABLENAME.

`@@VARIABLENAME@@` is replaced by the global variable called VARIABLENAME.

`@_VARIABLENAME_@` is replaced by the catalog variable VARIABLENAME if it exists; otherwise, it is replaced by the global variable VARIABLENAME.

For more information on how to use the `Variable` configuration file directive to set global variables in `interchange.cfg` and catalog variables in `catalog.cfg`, see the *Interchange Configuration Guide*.

3. Using Interchange Template Tags

This section describes the different template specific tags and functions that are used when building a your templates.

3.1. Understanding Tag Syntax

Interchange uses a style similar to HTML, but with [square brackets] replacing <chevrons>. The parameters that can be passed are similar, where a parameter="parameter value" can be passed.

Summary:

```
[tag parameter]             Tag called with positional parameter
[tag parameter=value]       Tag called with named parameter
[tag parameter="the value"] Tag called with space in parameter
[tag 1 2 3]                 Tag called with multiple positional parameters
[tag foo=1 bar=2 baz=3]     Tag called with multiple named parameters
[tag foo=`2 + 2`]           Tag called with calculated parameter
[tag foo="[value bar]"]     Tag called with tag inside parameter
[tag foo="[value bar]"]
    Container text.         Container tag.
[/tag]
```

Most tags can accept some positional parameters. This makes parsing faster and is, in most cases, simpler to write.

The following is an example tag:

```
[value name=city]
```

This tag causes Interchange to look in the user form value array and return the value of the form parameter `city`, which might have been set with:

```
City: <INPUT TYPE=text NAME=city VALUE="[value city]">
```

Note: Keep in mind that the value was pre−set with the value of `city` (if any). It uses the positional style, meaning `name` is the first positional parameter for the [value ...] tag. Positional parameters cannot be derived from other Interchange tags. For example, [value [value formfield]] will not work. But, if the named parameter syntax is used, parameters can contain other tags. For example:

```
[value name="[value formfield]"]]
```

There are exceptions to the above rule when using list tags such as [item−list], [loop], [query] and others. These tags, and their exceptions, are explained in their corresponding sections.

Many Interchange tags are container tags. For example:

```
[set Checkout]
    mv_nextpage=ord/checkout
    mv_todo=return
[/set]
```

Tags and parameter names are not case sensitive, so `[VALUE NAME=something]` and `[value name=something]` work the same. The Interchange development convention is to type HTML tags in upper case and Interchange tags in lower case. This makes pages and tags easier to read.

Single quotes work the same as double quotes and can prevent confusion. For example:

```
[value name=b_city set='[value city]']
```

Backticks should be used with extreme caution since they cause the parameter contents to be evaluated as Perl code using the [calc] tag. For example:

```
[value name=row_value set=`$row_value += 1`]
```

is the same as

```
[value name=row_value set="[calc]$row_value += 1[/calc]"]
```

Vertical bars can also be used as quoting characters, but have the unique behavior of stripping leading and trailing whitespace. For example:

```
[loop list="
    k1    A1    A2    A3
    k2    B1    B2    B3"]
    [loop-increment][loop-code]
[/loop]
```

could be better expressed as:

```
[loop list=|
        k1    A1    A2    A3
        k2    B1    B2    B3
|]
    [loop-increment][loop-code]
[/loop]
```

How the result of the tag is displayed depends on if it is a container or a standalone tag. A container tag has a closing tag (for example, `[tag] stuff [/tag]`). A standalone tag has no end tag (for example, [area href=somepage]. Note that [page] and [order] are **not** container tags. (`[/page]` and `[/order]` are simple macros.)

A container tag will have its output re−parsed for more Interchange tags by default. To inhibit this behavior, set the attribute reparse to 0. However, it has been found that the default re−parsing is almost always desirable. On the other hand, the output of a standalone tag will not be re−interpreted for Interchange tag constructs (with some exceptions, like [include file].

Most container tags will not have their contents interpreted (Interchange tags parsed) before being passed the container text. Exceptions include [calc], [currency] and [seti]. All tags accept the `interpolate=1` tag modifier, which causes the interpretation to take place.

3.2. data and field

The [data] and [field] tags access elements of Interchange databases. They are the form used outside of the iterating lists, and are used to do lookups when the table, column/field or key/row is conditional based on

a previous operation.

The following are equivalent for attribute names:

```
table --> base
col   --> field --> column
key   --> code  --> row
```

The [field] tag looks in any tables defined as `ProductFiles`, in that order, for the data and returns the first non−empty value. In most catalogs, where `ProductFiles` is not defined, i.e., the demo, `[field title 00−0011]` is equivalent to `[data products title 00−0011]`. For example, [field col=foo key=bar] will not display something from the table "category" because "category" is not in the directive ProductFiles or there are multiple ProductFiles and an earlier one has an entry for that key.

[data table column key]

named attributes: `[data base="database" field="field" key="key" value="value" op="increment]`
Returns the value of the field in any of the arbitrary databases or from the variable namespaces. If the option `increment=1` is present, the field will be automatically incremented with the value in value.
If a DBM−based database is to be modified, it must be flagged writable on the page calling the write tag. For example, use `[tag flag write]products[/tag]` to mark the `products` database writable.
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
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 via UserDB
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)
```

Databases will hide variables, so if a database is named "session," "scratch," or any of the other reserved names it won't be able to use the `[data ...]` tag to read them. Case is sensitive, so the database could be called "Session," but this is not recommended practice.

[field name code]

named attributes: `[field code="code" name="fieldname"]`
Expands into the value of the field name for the product as identified by code found by searching the products database. It will return the first entry found in the series of Product Files in the products database. If this needs to constrained to a particular table, use a `[data table col key]` call.

3.3. set, seti, tmp, tmpn scratch and scratchd

Scratch variables are maintained in the user session, which is separate from the form variable values set on HTML forms. Many things can be controlled with scratch variables, particularly search and order processing, the `mv_click` multiple variable setting facility and key Interchange conditions session URL display.

There are four tags that are used to set the scratch space: [`set variable`] value [/set], [`seti variable`] value [/seti], [`tmp variable`] value [/tmp], [`tmpn variable`] value [/tmpn] and and two tags for reading scratch space: [`scratch variable`] and [`scratchd variable`].

[`set variable`] value [/set]

Sets a scratchpad variable to a value.
Most of the mv_* variables that are used for search and order conditionals are in another namespace. They can be set through hidden fields in a form.
An order profile would be set with:

```
[set checkout]
name=required Please enter your name.
address=required No address entered.
[/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">
```

To do the same as `[set foo]bar[/set]` in embedded Perl:

```
[calc]$Scratch->{foo} = 'bar'; return;[/calc]
```

[`seti variable`] value [/seti]

The same as [`set`] except it interpolates the container text. The above is the same as:

```
[set name=variable interpolate=1] [value something] [/set]
```

[`tmp variable`] value [/tmp]

The same as [`seti`] except it does not persist.

[`tmpn variable`] value [/tmpn]

The same as [`set`] except it does not persist.

[`scratch variable`]

Returns the contents of a scratch variable to the page. `[scratch foo]` is the same as, but faster than:

```
[perl] $Scratch->{foo} [/perl]
```

[scratchd variable]

The same as [scratch variable] except it deletes the value after returning it. Same as [scratch foo] [set foo] [/set].

[if scratch name op* compare*] yes [else] no [/else][/if]

Tests a scratch variable. See the [if] tag documentation for more information.

3.4. loop

Loop lists can be used to construct arbitrary lists based on the contents of a database field, a search or other value (like a fixed list). Loop accepts a search parameter that will do one−click searches on a database table (or file).

To iterate over all keys in a table, use the idiom ([loop search="ra=yes/ml=9999"] [/loop]. ra=yes sets mv_return_all, which means "match everything". ml=9999 limits matches to that many records. If the text file for searching an Interchange DBM database is not used, set st=db (mv_searchtype).

When using st=db, returned keys may be affected by TableRestrict. Both can be sorted with [sort table:field:mod −start +number] modifiers. See sorting.

The Interchange Tags Reference has more information on the [loop] tag.

[loop item item item] LIST [/loop]

named attributes: [loop prefix=label* list="item item item"* search="se=whatever"*]
Returns a string consisting of the LIST, repeated for every item in a comma−separated or space−separated list. This tag works the same way as the [item−list] tag, except for order−item−specific values. It is 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 by using different prefixes:

```
[loop prefix=size list="Small Medium Large"]
    [loop prefix=color list="Red White Blue"]
        [color-code]-[size-code]<BR>
    [/loop]
    <P>
[/loop]
```

This will output:

```
Red-Small
White-Small
Blue-Small

Red-Medium
White-Medium
Blue-Medium
```

```
Red-Large
White-Large
Blue-Large
```

The search="args" parameter will return an arbitrary search, 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."

[if-loop-data table field] IF [else] ELSE [/else][/if-loop-data]

Outputs the IF if the `field` in the `table` is not empty or the ELSE (if any) otherwise.

Note: This tag does not nest with other `if-loop-data` tags.

[if-loop-field] IF [else] ELSE [/else][/if-loop-field]

Outputs the IF if the `field` in the `products` table is not empty or the ELSE (if any) otherwise.

Note: This tag does not nest with other `if-loop-field` tags.

[loop-alternate N] DIVISIBLE [else] NOT DIVISIBLE [/else][/loop-alternate]

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

```
[loop-alternate 2]EVEN[else]ODD[/else][/loop-alternate]
[loop-alternate 3]BY 3[else]NOT by 3[/else][/loop-alternate]
```

[/loop-alternate]

Terminates the alternation area.

[loop-change marker]

Same as `[item-change]`, but within loop lists.

[loop-code]

Evaluates to the first returned parameter for the current returned record.

[loop-data database fieldname]

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

[loop-description]

Evaluates to the product description for the current item. Returns the <Description Field> from the first products database where that item exists.

[loop-field fieldname]

The [loop-field]
The [loop-field] tag is special in that it looks in any of the tables defined as `ProductFiles`, in that order, for the data and returns the value only if that key is defined. In most catalogs, where `ProductFiles` is not defined `[loop-field title]` is equivalent to [loop-field products title].
Evaluates to the field name fieldname in the database for the current item.

[loop-increment]

Evaluates to the number of the item in the list. Used for numbering items in the list. Starts from one (1).

[loop-last]tags[/loop-last]

Evaluates the output of the ITL tags encased in the [loop-last] tags. If it evaluates to a numerical non-zero number (for example, 1, 23, -10 etc.), the loop iteration will terminate. If the evaluated number is negative, the item itself will be skipped. If the evaluated number is positive, 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, 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 it will be the last item on the list.

[loop-next]tags[/loop-next]

Evaluates the output of the ITL tags encased in the [loop-next] tags. If it evaluates to a numerical non-zero number (for example, 1, 23, -10 etc.), 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, a numerical `1` will be output from the `[calc][/calc]` operation. The item will not be shown.

[loop-price n* noformat*]

Evaluates to the price for the 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.

[loop-calc] PERL [/loop-calc]

Calls embedded Perl with the code in the container. All `[loop-*]` tags can be placed inside except for `[loop-filter ...][/loop-filter]`, `[loop-exec routine][/loop-exec]`, `[loop-last][/loop-last]` and `[loop-next][/loop-next]`.

Note: All normal embedded Perl operations can be used, but be careful to pre−open any database tables with a [perl tables="tables you need"][/perl] tag prior to the opening of the [loop].

[loop−exec routine]argument[/loop−exec]

Calls a subroutine predefined either in catalog.cfg with Sub or in a [loop] with `[loop−sub routine]` PERL `[/loop−sub]`. The container text is passed as `$_[0]` and the array (or hash) value of the current row is `$_[1]`.

[loop−sub routine]PERL[/loop−sub]

Defines a subroutine that is available to the current (and subsequent) `[loop−*]` tags within the same page. See Interchange Programming.

3.5. if

[if type field op* compare*]

The Interchange Tags Reference has more information on the [if] tag.

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

[if !type field op* compare*]

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

Allows the 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 text 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 text will be output for a false condition.
[/then]
[elsif type term op compare]
                            Optional, tested when if fails.
[/elsif]
[else]
                            Optional, printed on the document when all above fail.
[/else]
[/if]
```

The [if] tag can also have some variants:

```
[if explicit]
[condition] CODE [/condition]
                            Displayed if valid Perl CODE returns a true value.
[/if]
```

Some Perl−style regular expressions can be written:

```
[if value name =~ /^mike/i]
                            This is the if with Mike.
[elsif value name =~ /^sally/i]
                            This is an elsif with Sally.
[/elsif]
[elsif value name =~ /^barb/i]
[or value name =~ /^mary/i]
                            This is an elsif with Barb or Mary.
[elsif value name =~ /^pat/i]
[and value othername =~ /^mike/i]
                            This is an elsif with Pat and Mike.
[/elsif]
[else]
                            This is the else, no name I know.
[/else]
[/if]
```

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

This will not work:

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

Do this instead:

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

As an alternative:

```
[if type=value term=high_water op="<" compare="[shipping noformat=1]"]
    The shipping cost is too high, charter a truck.
[/if]
```

There are many test targets available. The following is a list of some of the available test targets.

config Directive

The Interchange configuration variables. These are set by the directives in the Interchange configuration file.

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

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]
```

If another tag is needed to select the key, and it is not a looping tag construct, named parameters must be used:

```
[set code]99-102[/set]
[if type=data term="products::size::[scratch code]"]
There is size information.
[else]
No size information.
[/else]
[/if]
```

discount

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

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

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 = $ values =~{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]
```

The same thing could be accomplished with `[if value country =~ /u\.?s\.?a?/i]`, but there are many situations where this example could be useful.

file

Tests for the existence of a file. This is 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]

or

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

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

items

The Interchange shopping carts. If not specified, the cart used is the main cart. This is usually used to 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]
```

ordered

Order status of individual items in the Interchange shopping carts. Unless otherwise specified, the cart used is the main cart. The following items refer to a part number of 99-102.

[if ordered 99-102] ... [/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'. THE CART NAME IS REQUIRED IN THE OLD SYNTAX. The new syntax for that one would be:

```
[if type=ordered term="99-102" compare="size =~ /large/i"]
```

To make sure it is the size that is large and not another attribute, you could use:

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

[if ordered 99-102 main lines] ... [/if]
Special case -- counts the lines that the item code is present on. (Only useful, of course, when mv_separate_items or SeparateItems is defined.)

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]
```

session

The Interchange session variables. Of particular interest are logged_in, source, browser and username.

validcc

A special case, it 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. It performs a LUHN−10 calculation to weed out typos or phony card numbers.

value

The Interchange user variables, typically set in search, control or order forms. Variables beginning with `mv_` are Interchange special values and should be tested and used with caution.

variable

See Interchange *Variable* values.

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

As an example, consider buttonbars for frame−based setups. You might decide to display a different buttonbar with no frame targets for sessions that are not using frames:

```
[if session frames]
    [buttonbar 1]
[else]
    [buttonbar 2]
[/else]
[/if]
```

Another example might be the when search matches are displayed. If using the string `[value mv_match_count] titles found`, it will display a plural result even if there is 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 the same as they are 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 should be done with `[if explicit]`.

[then] text [/then]

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

[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.

[else] text [/else]

The optional else−text for an if or if−item−field conditional.

[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 in the [`perl`] construct.

[/if]

Terminates an if conditional.

4. Programming

Interchange has a powerful paradigm for extending and enhancing its functionality. It uses two mechanisms, user−defined tags and user subroutines on two different security levels, global and catalog. In addition, embedded Perl code can be used to build functionality into pages.

User−defined tags are defined with the UserTag directive in either `interchange.cfg` or `catalog.cfg`. The tags in `interchange.cfg` are global and they are not constrained by the `Safe` Perl module as to which opcodes and routines they may use. The user−defined tags in `catalog.cfg` are constrained by `Safe`. However, if the `AllowGlobal` global directive is set for the particular catalog in use, its UserTag and `Sub` definitions will have global capability.

4.1. Overriding Interchange Routines

Many of the internal Interchange routines can be accessed by programmers who can read the source and find entry points. Also, many internal Interchange routines can be overridden:

```
GlobalSub <<EOS
sub just_for_overriding {
    package Vend::Module;
    use MyModule;
    sub to_override {
        &MyModule::do_something_funky($Values->{my_variable});
    }
}
EOS
```

The effect of the above code is to override the `to_override` routine in the module `Vend::Module`. This is preferable to hacking the code for functionality changes that are not expected to change frequently. In most cases, updating the Interchange code will not affect the overridden code.

Note: Internal entry points are not guaranteed to exist in future versions of Interchange.

4.2. Embedding Perl Code

Perl code can be directly embedded in Interchange pages. The code is specified as:

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

ASP syntax can be used with:

```
[mvasp]
    <%
    $name    = $Values->{name};
    $browser = $Session->{browser};
    %>
    Hi, <%= $name %>!
    <%
```

```
        HTML "How do you like your $browser?";
    %>
[/mvasp]
```

The two examples above are essentially equivalent. See the perl and mvasp tags for usage details.

The [perl] tag enforces Safe.pm checking, so many standard Perl operators are not available. This prevents user access to all files and programs on the system without the Interchange daemon's permissions. See GlobalSub and User-defined Tags for ways to make external files and programs available to Interchange.

Named parameters:
See the perl tag for a description of the tag parameters and attributes. These include:

```
[perl tables="tables-to-open"*
        subs=1*
      global=1*
   no_return=1*
     failure="Return value in case of compile or runtime error"*
        file="include_file"*]
```

Required parameters: none

Any Interchange tag (except ones using SQL) can be accessed using the $Tag object. If using SQL queries inside a Perl element, AllowGlobal permissions are required and and the global=1 parameter must be set. Installing the module Safe::Hole along with sharing the database table with <tables=tablename> will enable SQL use.

For example:

```
# If the item might contain a single quote
[perl]
$comments = $Values->{comments};
[/perl]
```

Important Note: Global subroutines are not subject to the stringent security check from the Safe module. This means that the subroutine will be able to modify any variable in Interchange and will be able to write to read and write any file that the Interchange daemon has permission to write. Because of this, the subroutines should be used with caution. They are defined in the main interchange.cfg file and can't be reached by from individual users in a multi-catalog system.

Global subroutines are defined in interchange.cfg with the GlobalSub directive or in user catalogs which have been enabled through AllowGlobal. Catalog subroutines are defined in catalog.cfg, with the Sub directive and are subject to the stringent Safe.pm security restrictions that are controlled by the global directive SafeUntrap.

The code can be as complex as you want them to be, but cannot be used by operators that modify the file system or use unsafe operations like "system," "exec," or backticks. These constraints are enforced with the default permissions of the standard Perl module **Safe**. Operations may be untrapped on a system-wide basis with the SafeUntrap directive.

The result of this tag will be the result of the last expression evaluated, just as in a subroutine. If there is a syntax error or other problem with the code, there will be no output.

Here is a simple one which does the equivalent of the classic hello.pl program:

```
[perl] my $tmp = "Hello, world!"; $tmp; [/perl]
```

There is no need to set the variable. It is there only to show the capability.

To echo the user's browser, but within some HTML tags:

```
[perl]
my $html = '<H5>';
$html .= $Session->{browser};
$html .= '</H5>';
$html;
[/perl]
```

To show the user their name and the current time:

```
[perl]

my $string = "Hi, " . $Values->{name} ". The time is now ";
$string .= $Tag->time();
$string;

[/perl]
```

4.3. ASP−Like Perl

Interchange supports an ASP−like syntax using the [mvasp] tag.

```
[mvasp]
<HTML><BODY>
    This is HTML.<BR>

<% HTML "This is code<BR>"; %>
    More HTML.<BR>
<% $Document->write("Code again.<BR>") %>
[/mvasp]
```

If no closing [/mvasp] tag is present, the remainder of the page will also be seen as ASP.

ASP is simple. Anything between <% and %> is code, and the string %> can not occur anywhere inside. Anything not between those anchors is plain HTML that is placed unchanged on the page. Interchange variables, [L][/L] and [LC][/LC] areas will still be inserted, but any Interchange tags will not.

There is a shorthand <% = $foo %>, which is equivalent to <% $Document->write($foo); %> or <% HTML $foo; %>

```
[mvasp]
<HTML><BODY>
    This is HTML.<BR>
    [value name] will show up as &#91;value name].<BR>

    &#95_VARIABLE__ value is equal to: __VARIABLE__

<% = "This is code<BR>" %>
```

The __VARIABLE__ will be replaced by the value of `Variable VARIABLE`, but [value name] will be shown unchanged.

Important Note: If using the `SQL::Statement` module, the catalog must be set to `AllowGlobal` in `interchange.cfg`. It will not work in "Safe" mode due to the limitations of object creation in Safe. Also, the `Safe::Hole` module must be installed to have SQL databases work in Safe mode.

4.4. Error Reporting

If your Perl code fails with a compile or runtime error, Interchange writes the error message from the Perl interpreter into the catalog's error log. This is usually '*catalog_root*/error.log'. Error messages do not appear on your web page as the return value of the Perl tag or routine.

You will not have direct access to the '`strict`' and '`warnings`' pragmas where Interchange runs your perl code under `Safe` (for example, within a [perl] or [mvasp] tag).

5. Interchange Perl Objects

Interchange gives you access to the power of Perl with the [perl], [calc] and [mvasp] tags. They all support the same set of Perl objects and variables.

5.1. A note about Safe

You can access all objects associated with the catalog and the user settings with opcode restrictions based on the standard Perl module `Safe.pm`. There are some unique things to know about programming with Interchange.

Under `Safe`, certain things cannot be used. For instance, the following can not be used when running Safe:

```
$variable = `cat file/contents`;
```

The backtick operator violates a number of the default Safe opcode restrictions. Also, direct file opens can not be used. For example:

```
open(SOMETHING, "something.txt")
    or die;
```

This will also cause a trap, and the code will fail to compile. However, equivalent Interchange routines can be used:

```
# This will work if your administrator doesn't have NoAbsolute set
$users = $Tag->file('/home/you/list');

# This will always work, file names are based in the catalog directory
$users = $Tag->file('userlist');
```

5.2. Standard objects and variables

The following is a list of Interchange Perl standard objects are:

$CGI

This is a hash reference to `%CGI::values`, the value of user variables as submitted in the current page/form. To get the value of a variable submitted as

```
<INPUT TYPE=hidden NAME=foo VALUE=bar>
```

use

```
[perl]
            return "Value of foo is $CGI->{foo}";
    [/perl]
```

Actually, you should not do that -- if someone sends you a value you should not output it willy-nilly for security reasons. Filter it first with the [filter] tag as accessed by the $Tag object:

```
[perl]
```

```
                my $val = $Tag->filter('encode_entities', $CGI->{foo});
                return "Value of foo is $val";
        [/perl]
```

Remember, multiple settings of the same variable are separated by a NULL character. To get the array value, use $CGI_array.

$CGI_array

This is a hash reference to `%CGI::values_array`, arrays containing the value or values of user variables as submitted in the current page/form. To get the value of a variable submitted as

```
<INPUT TYPE=hidden NAME=foo VALUE='bar'>
<INPUT TYPE=hidden NAME=foo VALUE='baz'>
```

use

```
<% = "The values of foo are", join (' and ', @{$CGI_array->{'foo'}}) %>
```

Remember, multiple settings of the same variable are separated by a NULL character. To get the array value, use $CGI_array.

$Carts

A reference to the shopping cart hash $Vend::Session->{carts}. The normal default cart is "main". A typical alias is $Items.
Shopping carts are an array of hash references. Here is an example of a session cart array containing a `main` and a `layaway` cart.

```
{
    'main' => [
                {
                    'code' => '00-0011',
                    'mv_ib' => 'products',
                    'quantity' => 1,
                    'size' => undef,
                    'color' => undef
                },
                {
                    'code' => '99-102',
                    'mv_ib' => 'products',
                    'quantity' => 2,
                    'size' => 'L',
                    'color' => 'BLUE'
                }
            ],
    'layaway' => [
                {
                    'code' => '00-341',
                    'mv_ib' => 'products',
                    'quantity' => 1,
                    'size' => undef,
                    'color' => undef
                }
            ]
}
```

In this cart array, $Carts->{main}[1]{code} is equal to `99-102`. Normally, it would be equivalent to $Items->[1]{code}.

$Config

A reference to the $Vend::Cfg array. This is normally used with a large amount of the Interchange source code, but for simple things use something like:

```
# Allow searching the User database this page only
$Config->{NoSearch} =~ s/\buserdb\b//;
```

Changes are not persistent -- they are reset upon the next page access.

%Db

A hash of databases shared with the `tables="foo"` parameter to the `[perl]` resp. `[mvasp]` tag calls. Once the database is shared, it is open and can be accessed by any of its methods. This will not work with SQL unless the `Safe::Hole` module is installed or AllowGlobal is set for the catalog.

NOTE: This object is not present and the below will not work with [calc].

To get a reference to a particular table, specify its hash element:

```
my $db = $Db{products};
```

The available methods are:

```
    # Key for a normal table with one primary key
    $key = 'foo';

    # Array reference key for a COMPOSITE_KEY table
    $composite_ary_key = ['foo','bar','buz'];

    # Alternate hash reference key for a COMPOSITE_KEY table
    $composite_hash_key = { key1 => 'foo', key2 => 'bar', key3 => 'buz'};

    # Alternate null-separated key for a COMPOSITE_KEY table
    $composite_nullsep_key = join "\0", 'foo','bar','buz';

    ### Any of the composite key types may be substitued
    ### when COMPOSITE_KEY table

# access an element of the table
$field = $db->field($key, $column);

# set an element of the table
$db->set_field($key, $column_name, $value);

# atomic increment of an element of the table
$db->inc_field($key, $column_name, 1);

# Return a complete hash of the database row (minus the key)
$hashref = $db->row_hash($key);

# Return some fields from a row
my @fields = qw/sku price description/;
```

```
@values = $db->get_slice($key, \@fields);

# Set some fields in a row (slice)
my $key = 'os28004';
my @fields = qw/price description/;
my @values = (5.95, "Ergo Roller");
$array_ref = $db->set_slice($key, \@fields, \@values);

# Alternate way to set slice
my $key = 'os28004';
my %fields = ( price => 5.95, description => "Ergo Roller");
$array_ref = $db->set_slice($key, \%fields);

# Perform a SQL query, returning an array of arrays
# (the equivalent of DBI $sth->fetchall_arrayref)
$ary = $db->query($sql);

# Same as above, except receive
# hash reference of pointers to field positions and
# array reference containing list of fields
my $sql = 'select * from products';
($ary, $index_hash, $name_ary) = $db->query($sql);
$fields_returned = join ",", @$name_ary;
$pointer_to_price = $index_hash->{price};

# Perform a SQL query, returning an array of hashes
$ary = $db->query({ sql => $sql, hashref => 1 });

# see if element of the table is numeric
$is_numeric = $db->numeric($column_name);

# Quote for SQL query purposes
$quoted = $db->quote($value, $column_name);

# Check configuration of the database
$delimiter = $db->config('DELIMITER');

# Find the names of the columns (not including the key)
@columns = $db->columns();
# Insert the key column name
unshift @columns, $db->config('KEY');

# See if a column is in the table
$is_a_column = defined $db->test_column($column_name);

# See if a row is in the table
$is_present = $db->record_exists($key);

# Create a subroutine to return a single column from the table
$sub = $db->field_accessor($column);
for (@keys) {
    push @values, $sub->($key);
}

# Create a subroutine to set a single column in the database
$sub = $db->field_settor($column);
for (@keys) {
    $sub->($key, $value);
}

# Create a subroutine to set a slice of the database
$sub = $db->row_settor(@columns);
```

```
for (@keys) {
    $sub->($key, @values);
}

# Return a complete array of the database (minus the key)
@values = $db->row($key);

# Delete a record/row from the table
$db->delete_record($key);
```

%Sql

A hash of SQL databases that you shared with the `[perl tables="foo"]` parameter to the tag call. It returns the DBI database handle, so operations like the following can be performed:

NOTE: This object is not present and the below will not work with [calc].

```
[perl products]
  my $dbh = $Sql{products}
      or return "Database not shared.";
  my $sth = $dbh->prepare('select * from products')
      or return "Couldn't open database.";
  $sth->execute();
  my @record;
  while(@record = $sth->fetchrow()) {
      foo();
  }
  $sth = $dbh->prepare('select * from othertable')
      or return "Couldn't open database.";
  $sth->execute();
  while(@record = $sth->fetchrow()) {
      bar();
  }
[/perl]
```

$DbSearch

A search object that will search a database without using the text file. It is the same as Interchange's `db` searchtype. Options are specified in a hash and passed to the object. All multiple-field options should be passed as array references. Before using the $DbSearch object, it must be told which table to search. For example, to use the table `foo`, it must have been shared with `[mvasp foo]`.
There are three search methods: `array`, `hash` and `list`.

```
array    Returns a reference to an array of arrays (best)
hash     Returns a reference to an array of hashes (slower)
list     Returns a reference to an array of tab-delimited lines
```

\Example:

```
$DbSearch->{table} = $Db{foo};

$search = {

        mv_searchspec => 'Mona Lisa',
        mv_search_field => [ 'title', 'artist', 'price' ],
        mv_return_fields    => [ 'title' ]

    };
```

```
my $ary = $DbSearch->array($search);

if(! scalar @$ary) {
    return HTML "No match.\n";
}

for(@$ary) {
```

$Document

This is an object which will allow you to write and manipulate the output of your embedded Perl. For instance, you can emulate a non−parsed−header program with:

```
[perl]
        $Document->hot(1);
        for(1 .. 20) {
                $Document->write("Counting to $_...<br>");
                $Document->write( " " x 4096);
                $Tag->sleep(1);
        }
        $Document->write("Finished counting!");
        return;
[/perl]
```

Note the write of 4096 spaces. Because Interchange's link program is parsed by default and your web server (and the link program) have buffers, you need to fill up the buffer to cause a write. You can do it without the extra padding if you set the link up as a non−parsed−header program −− see your web server documentation on how to do that.

There are several methods associated with $Document:

```
HTML $foo;                      # Append $foo to the write buffer array
$Document->write($foo);         # object call to append $foo to the write
                                # buffer array
$Document->insert($foo);        # Insert $foo to front of write buffer array
$Document->header($foo, $opt); # Append $foo to page header
$Document->send();              # Send write buffer array to output, done
                                # automatically upon end of ASP, clears buffer
                                # and invalidates $Document->header()
$Document->hot(1);              # Cause writes to send immediately
$Document->hot(0);              # Stop immediate send
@ary = $Document->review();     # Place contents of write buffer in @ary
$Document->replace(@ary)        # Replace contents of write buffer with @ary
$ary_ref = $Document->ref();    # Return ref to output buffer
```

$Document−>write($foo)

Write $foo to the page in a buffered fashion. The buffer is an array containing the results of all previous `$Document->write()` operations. If $Document−>hot(1) has been set, the output immediately goes to the user.

$Document−>insert($foo)

Insert $foo to the page buffer. The following example will output "123"

```
$Document->write("23");
```

```
$Document->insert("1");
$Document->send();
```

while this example will output "231"

```
$Document->write("23");
$Document->write("1");
$Document->send();
```

will output "231".

$Document->header($foo, $opt)

Add the header line $foo to the HTTP header. This is used to change the page content type, cache options or other attributes. The code below changes the content type (MIME type) to text/plain:

```
$Document->header("Content-type: text/plain");
```

There is an optional hash that can be sent with the only valid value being "replace." The code below scrubs all previous header lines:

```
$Document->header("Content-type: text/plain", { replace => 1 } );
```

Once output has been sent with $Document->send(), this can no longer be done.

$Document->hot($foo)

If the value of $foo is true (in a Perl sense), then all $Document->write() operations will be immediately sent until a $Document->hot(0) is executed.

$Document->send()

Causes the document write buffer to be sent to the browser and empties the buffer. Any further $Document->header() calls will be ignored. Can be used to implement non-parsed-header operation.

$Document->review()

Returns the value of the write buffer.

```
@ary = $Document->review();
```

$Document->replace(@new)

Completely replaces the write buffer with the arguments.

$Document->ref()

Returns a reference to the write buffer.

```
# Remove the first item in the write buffer
my $ary_ref = $Document->ref();
shift @$ary_ref;
```

HTML

Writes a string (or list of strings) to the write buffer array. The call

```
HTML $foo, $bar;
```

is exactly equivalent to

```
$Document->write($foo, $bar);
```

Honors the $Document->hot() setting.

$Items

A reference to the current shopping cart. Unless an Interchange `[cart ...]` tag is used, it is normally the same as $Carts->{main}.

$Scratch

A reference to the scratch values ala `[scratch foo]`.

```
<% $Scratch->{foo} = 'bar'; %>
```

is equivalent to:

```
[set foo]bar[/set]
```

$Session

A reference to the session values ala `[data session username]`.

```
<%
    my $out = $Session->{browser};
    $Document->write($out);
%>
```

is equivalent to:

```
[data session browser]
```

Values can also be set. If the value of `[data session source]` needed to be changed, for example, set:

```
<%
    $Session->{source} = 'New_partner';
%>
```

$Tag

Using the $Tag object, any Interchange tag including user-defined tags can be accessed.

IMPORTANT NOTE: If the tag will access a database that has not been previously opened, the table name must be passed in the ITL call. For example:

Named parameters:

```
[perl tables="products pricing"]
```

or
Positional parameters:

```
[perl products pricing]
```

Any tag can be called.

```
[perl]
    my $user = $Session->{username};
    return $Tag->data('userdb', 'name', $user );
[/perl]
```

is the same as:

```
[data table=userdb column=name key="[data session username]"]
```

If the tag has a dash (-) in it, use an underscore instead:

```
# WRONG!!!
$Tag->shipping-desc('upsg');
# Right
$Tag->shipping_desc('upsg');
```

There are two ways of specifying parameters. Either use the positional parameters as documented (for an authoritative look at the parameters, see the %Routine value in Vend::Parse) or specify it all with an option hash parameter names as in any named parameters as specified in an Interchange tag. The calls

```
$Tag->data('products', 'title', '00-0011');
```

and

```
my $opt = {
                table   => 'products',
                column  => 'title',
                key     => '00-0011',
            };

$Tag->data( $opt );
```

are equivalent for the `data` tag.
If using the option hash method, and the tag has container text, either specify it in the hash parameter `body` or add it as the next argument. The two calls:

```
$Tag->item_list( {
                    'body' => "[item-code] [item-field title]",
                });
```

and

```
$Tag->item_list( { }, "[item-code] [item-field title]")
```

are equivalent.
Parameter names are ALWAYS lower case.

$Values

A reference to the user form values ala `[value foo]`.

```
<% $Document->write($Values->{foo}); %>
```

is equivalent to:

```
[value foo]
```

&Log

Send a message to the error log (same as ::logError in GlobalSub or global UserTag).

```
<%
    Log("error log entry");
%>
```

It prepends the normal timestamp with user and page information. To suppress that information, begin the message with a backslash (\).

```
<%
    Log("\\error log entry without timestamp");
    Log('\another error log entry without timestamp');
    Log("error log entry with timestamp");
%>
```

6. Debugging

No debug output is provided by default. The source files contain commented−out '::logDebug(SOMETHING)' statements which can be edited to activate them. Set the value of `DebugFile` to a file that will be written to:

```
DebugFile /tmp/icdebug
```

6.1. Export

Named Parameters: [export table="dbtable"]
Positional Parameters: [export db_table]
The attribute hash reference is passed to the subroutine after the parameters as the last argument. This may mean that there are parameters not shown here. Must pass named parameter interpolate=1 to cause interpolation.
Invalidates cache: YES
Called Routine:
ASP/perl tag calls:

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

OR

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

Attribute aliases:

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

6.2. Time

Named Parameters: [time locale="loc"]
Positional Parameters: [time loc]
The attribute hash reference is passed after the parameters but before the container text argument. This may mean that there are parameters not shown here. Must pass named parameter interpolate=1 to cause interpolation.
This is a container tag, i.e., [time] FOO [/time].
Nesting: NO.
Invalidates cache: NO.
Called Routine:
ASP/perl tag calls:

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

OR

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

6.3. Import

Named Parameters: [import table=table_name type=(TAB|PIPE|CSV|%%|LINE)
continue=(NOTES|UNIX|DITTO) separator=c]
Positional Parameters: [import table_name TAB]
The attribute hash reference is passed after the parameters but before the container text argument. This may mean that there are parameters not shown here. Interpolates container text by default>.
This is a container tag, i.e., [import] FOO [/import].
Nesting: NO
Invalidates cache: YES.
Called Routine:
ASP/perl tag calls:

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

OR

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

Attribute aliases:

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

Description:
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. (Use SQL for on-the-fly tables.)
The type of LINE and continue setting of NOTES is particularly useful, for it allows the naming of fields so that the order in which they appear in the database will not have to be remembered. The following two imports are identical in effect:

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

[import table=orders type=LINE continue=NOTES]
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 NOTES mode is not used, 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, use a separator, which by default is a form−feed character (^L).

6.4. Log

Named Parameters: [log file=file_name]
Positional Parameters: [log file_name]
The attribute hash reference is passed after the parameters but before the container text argument. This may mean that there are parameters not shown here. Must pass named parameter interpolate=1 to cause interpolation. This is a container tag, i.e., [log] FOO [/log].
Nesting: NO.
Invalidates cache: NO.
Called Routine:
ASP/perl tag calls:

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

OR

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

Attribute aliases:

```
    arg ==> file
```

6.5. Header

6.6. price, description, accessories

[price code quantity* database* noformat*]

named attributes: `[price code="code" quantity="N" base="database" noformat=1* optionX="value"]`
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`.
If an named attribute corresponding to a product option is passed, and that option would cause a change in the price, the appropriate price will be displayed.
Demo example: The T−Shirt (product code 99−102), with a base price of $10.00, can vary in price depending on size and color. `S`, the small size, is 50 cents less; `XL`, the extra large size, is $1.00 more and the color `RED` is 0.75 extra. There are also quantity pricing breaks (see the demo `pricing` database. So the following will be true:

```
[price  code=99-102
        size=L]             is $10.00

[price  code=99-102
        size=XL]            is $11.00

[price  code=99-102
        color=RED
        size=XL]            is $11.75

[price  code=99-102
        size=XL
        quantity=10]        is $10.00

[price  code=99-102
        size=S]             is $9.50
```

An illustration of this is on the simple `flypage` template when passed that item code.

[description code table*]

named attributes: `[description code="code" base="database"]`
Expands into the description 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 table.

[accessories code attribute*, type*, field*, database*, name*, outboard*]

named attributes: `[accessories code="code" arg="attribute*, type*, field*, database*, name*, outboard*"]`
Initiates special processing of item attributes based on entries in the product database. See Item Attributes for a complete description of the arguments.
When called with an attribute, the database is consulted and looks for a comma-separated list of attribute options. They take the form:

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

The label text is optional. If none is given, the **name** will be used.
If an asterisk is the last character of the label text, the item is the default selection. If no default is specified, the first will be the default. An example:

```
[accessories TK112 color]
```

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

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

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

6.7. FILE and INCLUDE

These elements read a file from the disk and insert the contents in the location of the tag. `[include ...]` will allow insertion of Interchange variables and ITL tags.

[file ...]

named: [file name="name" type="dos|mac|unix"*]
positional: [file name]
Inserts the contents of the named file. The file should normally be relative to the catalog directory. File names beginning with / or .. are only allowed if the Interchange server administrator has disabled NoAbsolute. The optional `type` parameter will do an appropriate ASCII translation on the file before it is sent.

[include file]

named attributes: `[include file="name"]`
Same as `[file name]` except interpolates for all Interchange tags and variables.

6.8. Banner/Ad rotation

Interchange has a built−in banner rotation system designed to show ads or other messages according to category and an optional weighting.

The `[banner ...]` ITL tag is used to implement it.

The weighting system pre−builds banners in the directory 'Banners,' under the temporary directory. It will build one copy of the banner for every one weight. If one banner is weighted `7`, one `2` and one `1`, then a total of 10 pre−built banners will be made. The first will be displayed 70 percent of the time, the second 20 percent and the third 10 percent, in random fashion. If all banners need to be equal, give each a weight of 1.

Each category may have separate weighting. If the above is placed in category `tech`, then it will behave as above when placed in `[banner category=tech]` in the page. A separate category, say `art`, would have its own rotation and weighting.

The `[banner ...]` tag is based on a database table, named `banners` by default. It expects a total of five (5) fields in the table:

code

This is the key for the item. If the banners are not weighted, this should be a category specific code.

category

To choose to categorize weighted ads, this contains the category to select. If empty, it will be placed in the default (or blank) category.

weight

Must be an integer number 1 or greater to include this ad in the weighting. If 0 or blank, the ad will be ignored when weighted ads are built.

rotate

If the weighted banners are not used, this must contain some value. If the field is empty, the banner will not be displayed. If the value is specifically `0` (zero), then the entire contents of the `banner` field will be displayed when this category is used. If it is non−zero, then the contents of the `banner` field will be split into segments (by the separator `{or}`). For each segment, the banners will rotate in sequence for that user only. Obviously, the first banner in the sequence is more likely to be displayed than the last.
Summary of values of rotate field:

```
non−zero, non−blank: Rotating ads
blank:               Ad not displayed
0:                   Ad is entire contents of banner field
```

banner

This contains the banner text. If more than one banner is in the field, they should be separated by the text `{or}` (which will not be displayed).

Interchange expects the `banner` field to contains the banner text. It can contain more than one banner, separated by the string '{or}.' To activate the ad, place any string in the field `rotate`.

The special key "default" is the banner that is displayed if no banners are found. (Doesn't apply to weighted banners.)

Weighted banners are built the first time they are accessed after catalog reconfiguration. They will not be rebuilt until the catalog is reconfigured or the file tmp/Banners/total_weight and tmp/Banners/<category>/total_weight is removed.

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.

The database specification should make the `weight` field numeric so that the proper query can be made. Here is the example from Interchange's demo:

```
Database   banner   banner.txt   TAB
Database   banner   NUMERIC      weight
```

Examples:

weighted, categorized

To select categorized and weighted banners:
The `banner` table would look like this:

```
code    category   weight   rotate   banner
t1      tech       1                 Click here for a 10% banner
t2      tech       2                 Click here for a 20% banner
t3      tech       7                 Click here for a 70% banner
a1      art        1                 Click here for a 10% banner
a2      art        2                 Click here for a 20% banner
a3      art        7                 Click here for a 70% banner
```

Tag would be:

```
[banner weighted=1 category="tech"]
```

This will find *all* banners with a weight >= 1 where the `category` field is equal to `tech`. The files will be made into the director `tmp/Banners/tech`.

weighted

To select weighted banners:

```
[banner weighted=1]
```

This will find *all* banners with a weight >= 1. (Remember, integers only.) The files will be made into the director `tmp/Banners`.

```
code    category   weight   rotate   banner
t1      tech       1                 Tech banner 1
t2      tech       2                 Tech banner 2
t3      tech       7                 Tech banner 3
a1      art        1                 Art banner 1
a2      art        2                 Art banner 2
a3      art        7                 Art banner 3
```

Each of the above with a weight of 7 will actually be displayed 35 percent of the time.

categorized, not rotating

```
[banner category="tech"]
```

This is equivalent to:

```
[data table=banner col=banner key=tech
```

The differences are that it is not selected if "rotate" field is blank; if not selected, the default banner is displayed.
The `banner` table would look like this:

```
code    category   weight   rotate   banner
tech               0        0        Tech banner
```

Interchange tags can be inserted in the category parameter, if desired:

```
[banner category="[value interest]"]
```

categorized and rotating

```
[banner category="tech"]
```

The difference between this and above is the database.
The `banner` table would look like this:

```
code    category   weight   rotate   banner
tech               0        1        Tech banner 1{or}Tech banner 2
art                0        1        Art banner 1{or}Art banner 2
```

This would rotate between banner 1 and 2 for the category tech for each user. Banner 1 is always displayed first. The `art` banner would never be displayed unless the tag `[banner category=art]` was used, of course.
Interchange tags can be inserted in the category parameter, if desired:

```
[banner category="[value interest]"]
```

multi−level categorized

```
[banner category="tech:hw"] or [banner category="tech:sw"]
```

If have a colon−separated category, Interchange will select the most specific ad available. If the `banner` table looks like this:

```
code    category   weight   rotate   banner
tech               0        1        Tech banner 1{or}Tech banner 2
tech:hw            0        1        Hardware banner 1{or}HW banner 2
tech:sw            0        1        Software banner 1{or}SW banner 2
```

This works the same as single−level categories, except that the category tech:hw will select that banner. The category tech:sw will select its own. But, the category tech:html would just get the "tech" banner. Otherwise, it works just as in other categorized ads. Rotation will work if set non−zero/non−blank and it will be inactive if the rotate field is blank. Each category rotates on its own.

Advanced

All parameters are optional since they are marked with an asterisk `(*)`.
Tag syntax:

```
[banner
    weighted=1*
    category=category*
    once=1*
    separator=sep*
    delimiter=delim*
    table=banner_table*
    a_field=banner_field*
    w_field=weight_field*
    r_field=rotate_field*
]
```

Defaults are blank except:

```
table       banner    selects table used
a_field     banner    selects field for banner text
delimiter   {or}      delimiter for rotating ads
r_field     rotate    rotate field
separator   :         separator for multi−level categories
w_field     weight    rotate field
```

6.9. Tags for Summarizing Shopping Basket/Cart

The following elements are used to access common items which need to be displayed on baskets and checkout pages.

*** marks an optional parameter**

[item−list cart*]

named attributes: `[item−list name="cart"]`
Places an iterative list of the items in the specified shopping cart, the main cart by default. See Item Lists for a description.

[/item−list]

Terminates the `[item−list]` tag.

[nitems cart*]

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

[subtotal]

Expands into the subtotal cost, exclusive of sales tax, of all the items ordered so far.

[salestax cart*]

Expands into the sales tax on the subtotal of all the items ordered so far. If there is no key field to derive the proper percentage, such as state or zip code, it is set to 0. See SALES TAX for more information.

[shipping−description mode*]

named attributes: `[shipping−description name="mode"]`
The text description of mode. The default is the shipping mode currently selected.

[shipping mode*]

named attributes: `[shipping name="mode"]`
The shipping cost of the items in the basket via `mode`. The default mode is the shipping mode currently selected in the `mv_shipmode` variable. See SHIPPING.

[total−cost cart*]

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

[currency convert*]

named attributes: `[currency convert=1*]`
When passed a value of a single number, formats it according to the currency specification. For instance:

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

will display:

```
4.00
```

Uses the Locale and PriceCommas settings as appropriate, and can contain a [calc] region. If the optional "convert" parameter is set, it will convert according to PriceDivide> for the current locale. If Locale is set to fr_FR, and PriceDivide for fr_FR is 0.167, using the following sequence:

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

will cause the number 8.982,04 to be displayed.

[/currency]

Terminates the currency region.

[cart name]

named attributes: [cart name="name"]
Sets the name of the current shopping cart for display of [shipping], [price], [total], [subtotal] and [nitems] tags. If a different price is used for the cart, all of the above except [shipping] will reflect the normal price field. Those operations must be emulated with embedded Perl or the [item-list], [calc] and [currency] tags, or use the PriceAdjustment feature to set it.

[row nn]

named attributes: [row width="nn"]
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.

[/row]

Terminates a [row nn] element.

[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, including the gutter. Must be
                supplied, there is no default. A shorthand method
                is to just supply the number as the 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 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.
```

[/col]

Terminates the column field.

6.10. Item Lists

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:

[if−item−data table column]

If the database field `column` in table *table* is non−blank, the following text up to the `[/if−item−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.

Note: This tag does not nest with other `[if−item−data ...]` tags.

[if−item−data table column]

Reverses sense for `[if−item−data]`.

[/if−item−data]

Terminates an `[if−item−data table column]` element.

[if−item−field fieldname]

If the products database field fieldname is non−blank, the following text up to the `[/if−item−field]` tag is substituted. If there are 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.

Note: This tag does not nest with other `[if−item−field ...]` tags.

[if−item−field fieldname]

Reverses sense for `[if−item−field]`.

[/if−item−field]

Terminates an `[if−item−field fieldname]` element.

[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.

[item−alternate N] DIVISIBLE [else] NOT DIVISIBLE [/else][/item−alternate]

Sets 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, 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]
```

[/item−alternate]

Terminates the alternation area.

[item−code]

Evaluates to the product code for the current item.

[item−data database fieldname]

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

[item−description]

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

[item−field fieldname]

The [item−field ...] tag is special in that it looks in any of the tables defined as `ProductFiles`, in that order, for the data, returning the value only if that key is defined. In most catalogs, where `ProductFiles` is not defined (i.e., the demo), `[item−field title]` is equivalent to `[item−data products title]`.
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.

[item−increment]

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

[item−last]tags[/item−last]

Evaluates the output of the Interchange tags encased inside the tags. If it evaluates to a numerical non−zero number (i.e., 1, 23, or −1), the list iteration will terminate. If the evaluated number is negative, the item itself will be skipped. If the evaluated number is positive, 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 the `[item-list]` (or `[search-list]` or flypage) and the weight field is empty, 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: there is no equivalent HTML style.

[item-modifier attribute]

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

[item-next]tags[/item_next]

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

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

If this is contained in the `[item-list]` (or `[search-list]` or flypage) and the product's weight field is less than 1, 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.)

[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, currency formatting will not be applied.

[item-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, currency formatting will not be applied. Returns regular price if not discounted.

[item-discount]

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

[item-quantity]

Evaluates to the quantity ordered for the current item.

[item-subtotal]

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

[item-modifier-name attribute]

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

[quantity−name]

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

7. Interchange Page Display

Interchange has several methods for displaying pages:

- Display page by name
  If a page with `[page some_page]` or `<A HREF="[area some_page]">` is called and that `some_page.html` exists in the pages directory (`PageDir`), it will be displayed.
- On−the−fly page
  If a page with `[page 00−0011]` or `<A HREF="[area 00−0011]">` is called and `00−0011` exists as a product in one of the products databases (`ProductFiles`), Interchange will use the special page descriptor `flypage` as a template and build based on that part number. This is partly for convenience; the same thing can be accomplished by calling `[page your_template 00−0011]` and using the `[data session arg]` to perform the templating. But there is special logic associated with the `PageSelectField` configuration attribute to allow pages to be built with varying templates.
- Determine page via form action and variables
  If a form action, in almost all cases the page to display will be determined by the `mv_nextpage` form value. Example:

```
<FORM ACTION="[process]">
<INPUT TYPE=hidden NAME=mv_todo VALUE=return>
<SELECT NAME=mv_nextpage>
<OPTION VALUE=index>Main page
<OPTION VALUE=browse>Product listing
<OPTION VALUE="ord/basket">Shopping cart
</SELECT>
<INPUT TYPE=submit VALUE=Go>
</FORM>
```

The `mv_nextpage` dropdown will determine the page the user goes to.

7.1. On−the−fly Catalog Pages

If an item is displayed on the search list (or order list) and there is a link to a special page keyed on the item, Interchange will attempt to build the page "on the fly." It will look for the special page flypage.html, which is used as a template for building the page. If `[item−field fieldname]`, `[item−price]` and similar elements are used on the page, complex and information−packed pages can be built. The `[if−item−field fieldname]` HTML `[/if−item−field]` pair can be used to insert HTML only if there is a non−blank value in a particular field.

Important note: Because the tags are substituted globally on the page, `[item−*]` tags cannot be used on the default on−the−fly page. To use a [search−region] or [item−list] tag, change the default with the prefix parameter. Example:

```
[item−list prefix=cart]
[cart−code] −− title=[cart−data products title]
[/item−list]
```

To have an on−the−fly page mixed in reliably, use the idiom `[fly−list prefix=fly code="[data session arg]"]` `[/flylist]` pair.

[fly−list code="product_code" base="table"] ... [/fly−list]

Other parameters:

```
prefix=label     Allows [label-code], [label-description]
```

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

```
[fly-list code="[data session arg]"]
 (contents of flypage.html)
[/fly-list]
```

If placed 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 </a>
[page flypage2 00-0011] Another way to display the Mona Lisa </a>
```

If the directive PageSelectField is set to a valid product database field which contains a valid Interchange page name (relative to the catalog pages directory, without the .html suffix), it will be used to build the on-the-fly page.

Active tags in their order of interpolation:

```
[if-item-field field]    Tests for a non-empty, non-zero value in field
[if-item-data db field]  Tests for a non-empty, non-zero field in db
[item-code]              Product code of the displayed item
[item-accessories args]  Accessory information (see accessories)
[item-description]       Description field information
[item-price quantity*]   Product price (at quantity)
[item-field field]       Product database field
[item-data db field]     Database db entry for field
```

7.2. Special Pages

A number of HTML pages are special for Interchange operation. Typically, they are used to transmit error messages, status of search or order operations and other out of boundary conditions.

Note: The distributed demo does not use all of the default values.

The names of these pages can be set with the *SpecialPage* directive. The standard pages and their default locations:

canceled (special_pages/canceled.html)

The page displayed by Interchange when an order has been canceled by the user.

catalog (special_pages/catalog.html)

The main catalog page presented by Interchange when another page is not specified.

failed (special_pages/failed.html)

If the sendmail program could not be invoked to email the completed order, the failed.html page is displayed.

flypage (special_pages/flypage.html)

If the catalog page for an item was not found when its `[item−link]` is clicked, this page is used as a template to build an on−the−fly page. See On−the−fly Catalog Pages.

interact (special_pages/interact.html)

Displayed if an unexpected response was received from the browser, such as not getting expected fields from submitting a form. This would probably happen from typos in the html pages, but could be a browser bug.

missing (special_pages/missing.html)

This page is displayed if the URL from the browser specifies a page that does not have a matching .html file in the pages directory. This can happen if the customer saved a bookmark to a page that was later removed from the database, for example, or if there is a defect in the code.
Essentially this is the same as a 404 error in HTTP. To deliberately display a 404 error, just put this in special_pages/missing.html:

```
[tag op=header]Status: 404 missing[/tag]
```

noproduct (special_pages/noproduct.html)

This page is displayed if the URL from the browser specifies the ordering of a product code which is not in the products file.

order (ord/basket.html)

This page is displayed when the customer orders an item. It can contain any or all of the customer−entered values, but is commonly used as a status display (or "shopping basket").

search (results.html)

Contains the default output page for the search engine results. Also required is an input page, which can be the same as search.html or an additional page. By convention Interchange defines this as the page `results`.

```
SpecialPage   search   results
```

violation (special pages/violation.html)

Displayed if a security violation is noted, such as an attempt to access a page denied by an `access_gate`. See UserDB.

7.3. Checking Page HTML

Interchange allows debugging of page HTML with an external page checking program. Because leaving this enabled on a production system is potentially a very bad performance degradation, the program is set in a the global configuration file with the CheckHTML directive. To check a page for validity, set the global directive CheckHTML to the name of the program (don't do any output redirection). A good choice is the freely available program Weblint. It would be set in interchange.cfg with:

```
CheckHTML  /usr/local/bin/weblint −s −
```

Of course, the server must be restarted for it to be recognized. The full path to the program should be used. If having trouble, check it from the command line (as with all external programs called by Interchange).

Insert `[flag type=checkhtml][/tag]` at the top or bottom of pages to check and the output of the checker should be appended to the browser output as a comment, visible if the page or frame source are viewed. To do this occasionally, use a Variable setting:

```
Variable  CHECK_HTML    [flag type=checkhtml]
```

and place __CHECK_HTML__ in the pages. Then set the Variable to the empty string to disable it.

8. Forms and Interchange

Interchange uses HTML forms for many of its functions, including ordering, searching, updating account information and maintaining databases. Order operations possibly include ordering an item, selecting item size or other attributes and reading user information for payment and shipment. Search operations may also be triggered by a form.

Interchange supports file upload with the `multipart/form-data` type. The file is placed in memory and discarded if not accessed with the `[value-extended name=filevar file_contents=1]` tag or written with `[value-extended name=filevar outfile=your_file_name]`. See Extended Value Access and File Upload.

Interchange passes variables from page to page automatically. Every user session that is started by Interchange automatically creates a variable set for the user. As long as the user session is maintained, and does not expire, any variables you set on a form will be "remembered" in future sessions.

Don't use the prefix mv_ for your own variables. Interchange treats these specially and they may not behave as you wish. Use the mv_ variables only as they are documented.

Interchange does not unset variables it does not find on the current form. That means you can't expect a checkbox to become unchecked unless you explicitly reset it.

8.1. Special Form Fields

Interchange treats some form fields specially, to link to the search engine and provide more control over user presentation. It has a number of predefined variables, most of whose names are prefixed with `mv_` to prevent name clashes with your variables. It also uses a few variables which are post-fixed with integer digits; those are used to provide control in its iterating lists.

Most of these special fields begin with `mv_`, and include:

(O = order, S = search, C = control, A = all, X in scratch space)

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_check		A	Any form, sets multiple user variables after update
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_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_fail_form		A	Sets CGI values to use on failed profile check
mv_fail_href		A	Sets page to display on on failed profile check
mv_fail_zero		A	Forces zeroing of current form values on failed profile check
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_form_profile		A	Check form with Interchange profile
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
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_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_success_form		A	Sets CGI values to use on successful profile check
mv_success_href		A	Sets page to display on on successful profile check
mv_success_zero		A	Forces zeroing of current form values on successful profile 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)

8.2. Form Actions

Interchange form processing is based on an `action` and a `todo`. This can be gated with `mv_form_profile` to determine actions and form values based on a check for required values or other preconditions.

The predefined actions at the first level are:

```
process       process a todo
```

```
search        form-based search
scan          path-based search
order         order an item
```

Any action can be defined with `ActionMap`.

The `process` action has a second `todo` level called with `mv_todo` or `mv_doit`. The `mv_todo` takes preference over `mv_doit`, which can be used to set a default if no `mv_todo` is set.

The action can be specified with any of:

page name

Calling the page "search" will cause the search action. `process` will cause a form process action, etc. Examples:

```
<FORM ACTION="/cgi-bin/simple/search" METHOD=POST>
<INPUT NAME=mv_searchspec>
</FORM>
```

The above is a complete search in Interchange. It causes a simple text search of the default products database(s). Normally hard-coded paths are not used, but a Interchange tag can be used to specify it for portability:

```
<FORM ACTION="[area search]" METHOD=POST>
<INPUT NAME=mv_searchspec>
</FORM>
```

The tag `[process]` is often seen in Interchange forms. The above can be called equivalently with:

```
<FORM ACTION="[process]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_todo VALUE=search>
<INPUT NAME=mv_searchspec>
</FORM>
```

mv_action

Setting the special variable `mv_action` causes the page name to be ignored as the action source. The above forms can use this as a synonym:

```
<FORM ACTION="[area foo]" METHOD=post>
<INPUT TYPE=hidden NAME=mv_action VALUE=search>
<INPUT NAME=mv_searchspec>
</FORM>
```

The page name will be used to set `mv_nextpage`, if it is not otherwise defined. If `mv_nextpage` is present in the form, it will be ignored.

The second level `todo` for the `process` action has these defined by default:

```
back         Go to mv_nextpage, don't update variables
search   Trigger a search
submit   Submit a form for validation (and possibly a final order)
go       Go to mv_nextpage (same as return)
return   Go to mv_nextpage, update variables
```

```
set      Update a database table
refresh  Go to mv_orderpage|mv_nextpage and check for
         ordered items
cancel   Erase the user session
```

If a page name is defined as an action with `ActionMap` or use of Interchange's predefined action `process`, it will cause form processing. First level is setting the special page name `process`, or mv_action set to do a form `process`, the Interchange form can be used for any number of actions. The actions are mapped by the ActionMap directive in the catalog configuration file, and are selected on the form with either the mv_todo or mv_doit variables.

To set a default action for a `process` form, set the variable `mv_doit` as a hidden variable:

```
<INPUT TYPE=hidden NAME=mv_doit VALUE=refresh>
```

When the mv_todo value is not found, the refresh action defined in mv_doit will be used instead.

More on the defined actions:

back

Goes to the page in mv_nextpage. No user variable update.

cancel

All user information is erased, and the shopping cart is emptied. The user is then sent to mv_nextpage.

refresh

Checks for newly−ordered items in `mv_order_item`, looking for on−the−fly items if that is defined, then updates the shopping cart with any changed quantities or options. Finally updates the user variables and returns to the page defined in mv_orderpage or mv_nextpage (in that order of preference).

return

Updates the user variables and returns to the page defined in mv_nextpage.

search

The shopping cart and user variables are updated, then the form variables are interpreted and the search specification contained therein is dispatched to the search engine. Results are returned on the defined search page (set by mv_search_page or the search page directives).

submit

Submits the form for order processing. If no order profile is defined with the `mv_order_profile` variable, the order is checked to see if the current cart contains any items and the order is submitted.
If there is an order profile defined, the form will be checked against the definition in the order profile and submitted if the pragma &final is set to yes. If `&final` is set to no (the default) and the check succeeds, the user will be routed to the Interchange page defined in mv_successpage or mv_nextpage. If the check fails, the user will be routed to mv_failpage or mv_nextpage in that order.

8.3. Profile checking

Interchange can check forms for compliance with requirements. The mechanism uses a **profile**, which is set via the `mv_form_profile` or `mv_order_profile` variables.

8.3.1. mv_form_profile

The `mv_form_profile` checks happen before any other action. They are designed to prevent submission of a form if required parameters are not present.

You define the profiles either in scratch space or with files specified in the *OrderProfile* directive.

Specifications take the form of an order page variable (like name or address), followed by an equals sign and a check type. There are many provided check types, and custom ones can be defined in a user−specified file using the `CodeDef` configuration directive.

Standard check types:

required

A non−blank value is required in the user session space. It may have been submitted on a previous form.

mandatory

Must be non−blank, and must have been specified on this form, not a saved value from a previous form

phone

The field must look like a phone number, by a very loose specification allowing numbers from all countries

phone_us

Must have US phone number formatting, with area code

state

Must be a US state, including DC and Puerto Rico.

province

Must be a Canadian province or pre−1997 territory.

state_province

Must be a US state or Canadian province.

zip

Must have US postal code formatting, with optional ZIP+4. Also called by the alias `us_postcode`.

ca_postcode

Must have Canadian postal code formatting. Checks for a valid first letter.

postcode

Must have Canadian or US postal code formatting.

true

Field begins with **y**, **1**, or **t** (Yes, 1, or True) − not case sensitive

false

Field begins with **n**, **0**, or **f** (No, 0, or False) − not case sensitive

email

Rudimentary email address check, must have an '@' sign, a name, and a minimal domain

regex

One or more regular expressions (space−separated) to check against. To check that all submissions of the "foo" variable have "bar" at the beginning, do:

```
foo=regex ^bar
```

You can add an error message by putting it in quotes at the end:

```
foo=regex ^bar "You must have bar at the beginning of this"
```

You can require that the value **not** match the regex by preceding the regex with a **!** character (and no space afterwards):

```
foo=regex !^bar "You may not have bar at the beginning!"
```

length

A range of lengths you want the input to be:

```
foo=length 4−10
```

That will require `foo` be from 4 to 10 characters long.

unique

Tests to see that the value would be a unique key in a table:

```
foo=unique userdb Sorry, that username is already taken
```

filter

Runs the value through an Interchange filter and checks that the returned value is equal to the original value.

```
foo=filter entities Sorry, no HTML allowed
```

To check for all lower−case characters:

```
foo=filter lower Sorry, no uppercase characters
```

Also, there are pragmas that can be used to change behavior:

&charge

Perform a real−time charge operation. If set to any value but "custom", it will use Interchange's CyberCash routines. To set to something else, use the value "custom ROUTINE". The ROUTINE should be a GlobalSub which will cause the charge operation to occur −− if it returns non−blank, non−zero the profile will have succeeded. If it returns 0 or undef or blank, the profile will return failure.

&credit_card

Checks the mv_credit_card_* variables for validity. If set to "standard", it will use Interchange's `encrypt_standard_cc` routines. This destroys the CGI value of mv_credit_card_number −− if you don't want that to happen (perhaps to save it for sending to CyberCash) then add the word `keep` on the end. Example:

```
# Checks credit card number and destroys number after encryption
# The charge operation can never work

&credit_card=standard
&charge=custom authorizenet

# Checks credit card number and keeps number after encryption
# The charge operation can now work

&credit_card=standard keep
&charge=custom authorizenet
```

You can supply your own check routine with a GlobalSub:

```
&credit_card=check_cc
```

The `GlobalSub` check_cc will be used to check and encrypt the credit card number, and its return value will be used to determine profile success.

&fail

Sets the mv_nextpage value for a failed check.

```
&fail=page4
```

If the submit process succeeds, the user will be sent to the page `page4`.

&fatal

Set to '&fatal=yes' if an error should generate the error page.

&final

Set to '&final=yes' if a successful check should cause the order to be placed.

&update

Set to '&update=yes' if a successful check should cause the variable to be copied from the CGI space to the Values space. This is like [update values] except only for that variable.
This is typically used when using a `mv_form_profile` check so that a failing check will not cause all values to be reset to their former state upon returning to the form.

&return

Causes profile processing to terminate with either a success or failure depending on what follows. If it is non−blank and non−zero, the profile succeeds.

```
# Success :)
&return 1

# Failure :\
&return 0
```

Will ignore the &fatal pragma, but &final is still in effect if set.

&set

Set a user session variable to a value, i.e. `&set=mv_email [value email]`. This will not cause failure if blank or zero.

&setcheck

Set a user session variable to a value, i.e. `&set=mv_email [value email]`. This **will** cause failure if set to a blank or zero. It is usually placed at the end after a &fatal pragma would have caused the process to stop if there was an error −− can also be used to determine pass/fail based on a derived value, as it will cause failure if it evaluates to zero or a blank value.

&success

Sets the mv_nextpage value if the profile succeeds. Example:

```
&success=page5
```

If the submit process succeeds, the user will be sent to the page `page5`.

&update

Normally if an mv_form_profile check fails none of the form values are put in the user session, meaning that the `[value ...]` tag will not reflect what the user has submitted. If you set `&update=yes`, then as each variable is checked its value will be updated. Example:

```
        &update=yes
```

Even if the profile check fails, any variables checked before the &fatal=yes setting will be updated.

As an added measure of control, the specification is evaluated for the special Interchange tags to provide conditional setting of order parameters. With the `[perl] [/perl]` capability, quite complex checks can be done. Also, the name of the page to be displayed on an error can be set in the `mv_failpage` variable.

Error messages are set by appending the desired error message to the line containing the check:

city=required Please fill in your city.

This sets the value of the error associated with name, and can be displayed with the `error` tag:

```
        [error name=city show_error=1]
```

8.4. Profile examples

The following file specifies a simple check of formatted parameters:

```
name=required You must give us your name.
address=required Oops! No address.
city=required
state=required
zip=required
email=required
phone_day=phone_us XXX-XXX-XXXX phone-number for US or Canada
&fatal=yes
email=email Email address missing the domain?
&success=ord/shipping
```

The profile above only performs the &success directive if all of the previous checks have passed -- the &fatal=yes will stop processing after the check of the email address if any of the previous checks failed.

If you want to place multiple order profiles in the same file, separate them with __END__, which must be on a line by itself.

8.5. User defined check routines

User-defined check routines can be defined with CodeDef in a file included in the code/ directory, the same as a `UserTag`.

```
CodeDef foo  OrderCheck
CodeDef foo Routine <<EOR
sub {
       # $ref is to Vend::Session->{'values'} hash
       # $var is the passed name of the variable
       # $val is current value of checked variable
       my($ref, $var, $val) = @_;

    my($ref, $var, $val) = @_;

       if($val ne 'bar') {
               return (undef, $var, "The value of foo must be bar");
```

```
    }
    else {
        return (1, $var, '');
    }
}
EOF
```

Now you can specify in an order profile:

```
foo_variable=foo
```

Very elaborate checks are possible. The return value of the subroutine should be a three−element array, consisting of:

1. the pass/fail ('1' or 'undef') status of the check;
2. the name of the variable which was checked;
3. a standard error message for the failure, in case a custom one has not been specified in the order profile.

The latter two elements are used by the `[error]` tag for on−screen display of form errors. The checkout page of the Foundation demo includes examples of this.

8.6. One−click Multiple Variables

Interchange can set multiple variables with a single button or form control. First define the variable set (or profile, as in search and order profiles) inside a scratch variable:

```
[set Search by Category]
mv_search_field=category
mv_search_file=categories
mv_todo=search
[/set]
```

The special variable `mv_click` sets variables just as if they were put in on the form. It is controlled by a single button, as in:

```
<INPUT TYPE=submit NAME=mv_click VALUE="Search by Category">
```

When the user clicks the submit button, all three variables will take on the values defined in the "Search by Category" scratch variable. Set the scratch variable on the same form as the button is on. This is recommended for clarity. The `mv_click` variable will not be carried from form to form, it must be set on the form being submitted.

The special variable `mv_check` sets variables for the form actions <checkout, control, refresh, return, search,> and <submit>. This function operates after the values are set from the form, including the ones set by `mv_click`, and can be used to condition input to search routines or orders.

The variable sets can contain and be generated by most Interchange tags. The profile is interpolated for Interchange tags before being used. This may not always operate as expected. For instance, if the following was set:

```
[set check]
[cgi name=mv_todo set=bar hide=1]
```

```
mv_todo=search
[if cgi mv_todo eq 'search']
do something
[/if]
[/set]
```

The if condition is guaranteed to be false, because the tag interpretation takes place before the evaluation of the variable setting.

Any setting of variables already containing a value will overwrite the variable. To build sets of fields (as in mv_search_field and mv_return_fields), comma separation if that is supported for the field must be used.

It is very convenient to use `mv_click` as a trigger for embedded Perl:

```
<FORM ...
<INPUT TYPE=hidden NAME=mv_check VALUE="Invalid Input">
...
</FORM>

[set Invalid Input]
[perl]
my $type        = $CGI->{mv_searchtype};
my $spell_check = $CGI->{mv_spelling_errors};
my $out = '';
if($spell_check and $type eq 'text') {
    $CGI->{mv_todo}     = 'return';
    $CGI->{mv_nextpage} = 'special/cannot_spell_check';
}
return;
[/perl]
[/set]
```

8.7. Checks and Selections

A "memory" for drop−down menus, radio buttons and checkboxes can be provided with the `[checked]` and `[selected]` tags.

[checked var_name value]

named attributes: `[checked name="var_name" value="value" cgi=0|1 multiple=0|1 default=0|1 case=0|1]`
This will output CHECKED if the variable `var_name` is equal to `value`. Set the `cgi` attribute to use cgi instead of values data. Not case sensitive unless `case` is set.
If the `multiple` attribute is defined and set to a non−zero value (1 is implicit) and if the value matches on a word/non−word boundary, it will be CHECKED. If the `default` attribute is set to a non−zero value, the box will be checked if the variable `var_name` is empty or zero.

[selected var_name value]

named attributes: `[selected name="var_name" value="value" cgi=0|1 multiple=0|1 default=0|1 case=0|1]`
This will output SELECTED if the variable `var_name` is equal to `value`. Set the `cgi` attribute to use cgi instead of values data. Not case sensitive unless `case` is set.
If the `multiple` argument is present, it will look for any of a variety of values. If the `default` attribute is

set, SELECT will be output if the variable is empty or zero. Not case sensitive unless `case` is set. Here is a drop−down menu that remembers an item−modifier color selection:

```
<SELECT NAME="color">
<OPTION [selected name=color value=blue]> Blue
<OPTION [selected name=color value=green]> Green
<OPTION [selected name=color value=red]> Red
</SELECT>
```

For databases or large lists of items, sometimes it is easier to use `[loop list="foo bar"]` and its `option` parameter. The above can be achieved with:

```
<SELECT NAME=color>
[loop list="Blue Green Red" option=color]
<OPTION> [loop−code]
[/loop]
</SELECT>
```

See also the `ictags` documentation on the [loop] tag.

8.8. Integrated Image Maps

Imagemaps can also be defined on forms, with the special form variable `mv_todo.map`. A series of map actions can be defined. The action specified in the *default* entry will be applied if none of the other coordinates match. The image is specified with a standard HTML 2.0 form field of type IMAGE. Here is an example:

```
<INPUT TYPE=hidden NAME="mv_todo.map" VALUE="rect submit 0,0 100,20">
<INPUT TYPE=hidden NAME="mv_todo.map" VALUE="rect cancel 290,2 342,18">
<INPUT TYPE=hidden NAME="mv_todo.map" VALUE="default refresh">
<INPUT TYPE=image  NAME="mv_todo" SRC="url_of_image">
```

All of the actions will be combined together into one image map with NCSA−style functionality (see the NCSA imagemap documentation for details), except that Interchange form actions are defined instead of URLs.

8.9. Setting Form Security

You can cause a form to be submitted securely (to the base URL in the SecureURL directive, that is) by specifying your form input to be ACTION="[process secure=1]".

To submit a form to the regular non−secure server, just omit the `secure` modifier.

8.10. Stacking Variables on the Form

Many Interchange variables can be "stacked," meaning they can have multiple values for the same variable name. As an example, to allow the user to order multiple items with one click, set up a form like this:

```
<FORM METHOD=POST ACTION="[process]">
<input type=checkbox name="mv_order_item" value="M3243"> Item M3243
<input type=checkbox name="mv_order_item" value="M3244"> Item M3244
<input type=checkbox name="mv_order_item" value="M3245"> Item M3245
<input type=hidden name="mv_doit" value="refresh">
```

```
<input type=submit name="mv_junk" value="Order Checked Items">
</FORM>
```

The stackable `mv_order_item` variable with be decoded with multiple values, causing the order of any items that are checked.

To place a "delete" checkbox on the shopping basket display:

```
<FORM METHOD=POST ACTION="[process]">
[item-list]
  <input type=checkbox name="[quantity-name]" value="0"> Delete
  Part number: [item-code]
  Quantity: <input type=text name="[quantity-name]" value="[item-quantity]">
  Description: [item-description]
[/item-list]
<input type=hidden name="mv_doit" value="refresh">
<input type=submit name="mv_junk" value="Order Checked Items">
</FORM>
```

In this case, first instance of the variable name set by `[quantity-name]` will be used as the order quantity, deleting the item from the form.

Of course, not all variables are stackable. Check the documentation for which ones can be stacked or experiment.

8.11. Extended Value Access and File Upload

Interchange has a facility for greater control over the display of form variables; it also can parse `multipart/form-data` forms for file upload.

File upload is simple. Define a form like:

```
<FORM ACTION="[process-target] METHOD=POST ENCTYPE="multipart/form-data">
<INPUT TYPE=hidden NAME=mv_todo     VALUE=return>
<INPUT TYPE=hidden NAME=mv_nextpage VALUE=test>
<INPUT TYPE=file NAME=newfile>
<INPUT TYPE=submit VALUE="Go!">
</FORM>
```

The [value-extended ...] tag performs the fetch and storage of the file. If the following is on the `test.html` page (as specified with `mv_nextpage` and used with the above form, it will write the file specified:

```
<PRE>
Uploaded file name: [value-extended name=newfile]
Is newfile a file? [value-extended name=newfile yes=Yes no=No test=isfile]

Write the file. [value-extended name=newfile outfile=junk.upload]
Write again with
 indication: [value-extended name=newfile
                             outfile=junk.upload
                             yes="Written."]
                             no=FAILED]

And the file contents:
[value-extended name=newfile file_contents=1]
</PRE>
```

The [value−extended] tag also allows access to the array values of stacked variables. Use the following form:

```
<FORM ACTION="[process−target] METHOD=POST ENCTYPE="multipart/form−data">
<INPUT TYPE=hidden NAME=testvar VALUE="value0">
<INPUT TYPE=hidden NAME=testvar VALUE="value1">
<INPUT TYPE=hidden NAME=testvar VALUE="value2">
<INPUT TYPE=submit VALUE="Go!">
</FORM>
```

and page:

```
testvar element 0: [value−extended name=testvar index=0]
testvar element 1: [value−extended name=testvar index=1]
testvar elements:
 joined with a space:   |[value−extended name=testvar]|
 joined with a newline: |[value−extended
                            joiner="\n"
                            name=testvar
                            index="*"]|
 first two only:    |[value−extended
                            name=testvar
                            index="0..1"]|
 first and last:    |[value−extended
                            name=testvar
                            index="0,2"]|
```

to observe this in action.

The syntax for `[value−extended ...]` is:

```
named: [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*]
```

positional: [value−extended name]

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`.

name

The form variable NAME. If no other parameters are present, the value of the variable will be returned. If there are multiple elements, by default they will all be returned joined by a space. If `joiner` is present, 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.

joiner

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

test

There are 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.

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 `*`, it will return all (joined by `joiner`). If a range, such as `0 .. 2`, it will return multiple elements.

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, it returns nothing.

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 images or other files are to be written to go to HTML space, use the HTTP server's `Alias` facilities or make a symbolic link.

ascii

To do an auto−ASCII translation before writing the `outfile`, set the `ascii` parameter to a non−blank, non−zero value. The default is no translation.

yes

The value that will be returned if a test is true or a file is written successfully. It defaults to `1` for tests and the empty string for uploads.

no

The value that will be returned if a test is false or a file write fails. It defaults to the empty string.

8.12. Updating Interchange Database Tables with a Form

Any Interchange database can be updated with a form using the following method. The Interchange user interface uses this facility extensively.

Note: All operations are performed on the database, not the ASCII source file. An [export table_name] operation will have to be performed for the ASCII source file to reflect the results of the update. Records in any database may be inserted or updated with the [query] tag, but form−based updates or inserts may also be performed.

In an update form, special Interchange variables are used to select the database parameters:

mv_data_enable (scratch)

\IMPORTANT: This must be set to a non−zero, non−blank value in the scratch space to allow data set functions. Usually it is put in an mv_click that precedes the data set function. For example:

```
[set update_database]
[if type=data term="userdb::trusted::[data session username]"]
    [set mv_data_enable]1[/set]
[else]
    [set mv_data_enable]0[/set]
[/else]
[/if]
[/set]
<INPUT TYPE=hidden NAME=mv_click VALUE=update_database>
```

mv_data_table

The table to update.

mv_data_key

The field that is the primary key in the table. It must match the existing database definition.

mv_data_function

UPDATE, INSERT or DELETE. The variable `mv_data_verify` must be set true on the form for a DELETE to occur.

mv_data_verify

Confirms a DELETE.

mv_data_fields

Fields from the form which should be inserted or updated. Must be existing columns in the table in question.

mv_update_empty

Normally a variable that is blank will not replace the field. If `mv_update_empty` is set to true, a blank value will erase the field in the database.

mv_data_filter_(field)

Instantiates a filter for `(field)`, using any of the defined Interchange filters. For example, if `mv_data_filter_foo` is set to `digits`, only digits will be passed into the database field during the set operation. A common value might be "entities", which protects any HTML by translating < into `<`, " into `&quot;`, etc.

The Interchange action set causes the update. Here are a pair of example forms. One is used to set the key to access the record (careful with the name, this one goes into the user session values). The second actually

performs the update. It uses the `[loop]` tag with only one value to place default/existing values in the form based on the input from the first form:

```
   <FORM METHOD=POST ACTION="[process]">
    <INPUT TYPE=HIDDEN name="mv_doit" value="return">
    <INPUT TYPE=HIDDEN name="mv_nextpage" value="update_proj">
    Sales Order Number <INPUT TYPE=TEXT SIZE=8
                            NAME="update_code"
                            VALUE="[value update_code]">
    <INPUT TYPE=SUBMIT name="mv_submit"  Value="Select">
    </FORM>
<FORM METHOD=POST ACTION="[process]">
   <INPUT TYPE=HIDDEN NAME="mv_data_table"    VALUE="ship_status">
   <INPUT TYPE=HIDDEN NAME="mv_data_key"      VALUE="code">
   <INPUT TYPE=HIDDEN NAME="mv_data_function" VALUE="update">
   <INPUT TYPE=HIDDEN NAME="mv_nextpage"      VALUE="updated">
   <INPUT TYPE=HIDDEN NAME="mv_data_fields"
               VALUE="code,custid,comments,status">
   <PRE>

   [loop arg="[value update_code]"]
   Sales Order <INPUT TYPE=TEXT NAME="code    SIZE=10 VALUE="[loop-code]">
  Customer No. <INPUT TYPE=TEXT NAME="custid" SIZE=30
                   VALUE="[loop-field custid]">
      Comments <INPUT TYPE=TEXT NAME="comments"
                   SIZE=30 VALUE="[loop-field comments]">
        Status <INPUT TYPE=TEXT NAME="status"
                   SIZE=10 VALUE="[loop-field status]">
   [/loop]
   </PRE>

       <INPUT TYPE=hidden NAME="mv_todo" VALUE="set">
       <INPUT TYPE=submit VALUE="Update table">
   </FORM>
```

The variables in the form do not update the user's session values, so they can correspond to database field names without fear of corrupting the user session.

8.12.1. Can I use Interchange with my existing static catalog pages?

Yes, but you probably won't want to in the long run. Interchange is designed to build pages based on templates from a database. If all you want is a shopping cart, you can mix standard static pages with Interchange, but it is not as convenient and doesn't take advantage of the many dynamic features Interchange offers.

That being said, all you usually have to do to place an order link on a page is:

```
   <A HREF="/cgi-bin/construct/order?mv_order_item=SKU_OF_ITEM">Order!</A>
```

Replace `/cgi-bin/construct` with the path to your Interchange link.