Interchange Documentation (Full)

Advanced Interchange Topics

1. Advanced Interchange Topics

- Maintaining production Interchange servers
- Interchange Administration Tool Development
- Making catalog skeletons for use with makecat
- Building custom link programs

2. Maintaining Interchange

Some utilities are supplied in the VendRoot/bin directory:

```
compile_link Compiles an Interchange vlink or tlink CGI link
configdump   Dumps the configuration directives for a particular catalog
dump         Dumps the session file for a particular catalog
expire       Expires sessions for a particular catalog
expireall    Expires all catalogs
makecat      Make catalog
```

Some example scripts for other functions are in the `eg/` directory of the software distribution.

Some thought should be given to where the databases, error logs, and session files should be located, especially on an ISP that might have multiple users sharing an Interchange server. In particular, put all of the session files and logs in a directory that is not writable by the user. This eliminates the possibility that the catalog may crash if the directory or file is corrupted.

To test the format of user catalog configuration files before restarting the server, set (from VendRoot):

```
bin/interchange −test
```

This will check all configuration files for syntax errors, which might otherwise prevent a catalog from booting. Once a catalog configures properly, user reconfiguration will not crash it. It will just cause an error. But, it must come up when the server is started.

2.1. Starting, Stopping, and Re−starting the Servers

The following commands need to have VENDROOT changed to the main directory where Interchange is installed. If the Interchange base directory is `/home/interchange/`, the start command would be `/home/interchange/bin/interchange`.

Do a `perldoc VENDROOT/bin/interchange` for full documentation.

To start the server with default settings:

```
VENDROOT/bin/interchange
```

Assuming the server starts correctly, the names of catalogs as they are configured will be displayed, along with a message stating the process ID it is running under.

It is usually best to issue a restart instead. It doesn't hurt to do a restart if you're actually starting the first time. And, if a server is already running (from this VENDROOT), a new start attempt will fail. To restart the server:

```
VENDROOT/bin/interchange −restart
```

The `−r` option is the same as `−restart`.

This is typically done to force Interchange to re−read its configuration. A message will be displayed stating that a TERM signal has been sent to the process ID the servers are running under. This information is also sent to `VENDROOT/error.log`. Check the error.log file for confirmation that the server has restarted

properly.

To stop the server:

```
VENDROOT/bin/interchange −stop
```

A message will be displayed stating that a TERM signal has been sent to the process ID the server is running under. This information is also sent to `VENDROOT/error.log`.

Because processes waiting for selection on some operating systems block signals, they may have to wait for HouseKeeping seconds to stop. The default is 60.

To terminate the Interchange server with prejudice, in the event it will not stop:

```
VENDROOT/bin/interchange −kill
```

2.2. UNIX and INET modes

Both UNIX−domain and INET−domain sockets can be used for communication. INET domain sockets are useful when more than one web server, connected via a local−area network (LAN), is used for accessing an Interchange server.

Important note: When sending sensitive information like credit card numbers over a network, always ensure that the data is secured by a firewall, or that the Interchange server runs on the same machine as any SSL−based server used for encryption.

Use the −i and −u flags if you only want to use one communication method:

```
# Start only in UNIX mode
VENDROOT/bin/interchange −r −u

# Start only in INET mode
VENDROOT/bin/interchange −r −i
```

2.3. User Reconfiguration

The individual catalogs can be reconfigured by the user by running the [reconfig] support tag. This should be protected by one of the several forms of Interchange authentication, preferably by HTTP basic authorization. See `RemoteUser`.

The command line can be reconfigured (as the Interchange user) with:

```
VENDROOT/bin/interchange −reconfig <catalog>
```

It is easy for the administrator to manually reconfigure a catalog. Interchange simply looks for a file `etc/reconfig` (based in the Interchange software directory) at HouseKeeping time. If it finds a script name that matches one of the catalogs, it will reconfigure that catalog.

2.4. Expiring Sessions

If Interchange is using DBM capability to store the sessions, periodically expire old sessions to keep the session database file from growing too large.

```
expire -c catalog
```

There is also an `expireall` script which reads all catalog entries in `interchange.cfg` and runs `expire` on them. The `expire` script accepts a `-r` option which tells it to recover lost disk space.

On a UNIX server, add a crontab entry such as the following:

```
# once a day at 4:40 am
40 4 * * *    perl /home/interchange/bin/expireall -r
```

Interchange will wait until the current transaction is finished before expiring, so this can be done at any time without disabling web access. Any search paging files for the affected session (kept in `ScratchDir`) will be removed as well.

If not running DBM sessions, use a Perl script to delete all files not modified in the last one or two days. The following will work if given an argument of a session directory or session files:

```
#!perl
# expire_sessions.pl -- delete files 2 days old or older

my @files;
my $dir;
foreach $dir (@ARGV) {
    # just push files on the list
    if (-f $dir) { push @files, $_; next; }

    next unless -d $dir;

    # get all the file names in the directory
    opendir DIR, $dir or die "opendir $dir: $!\n";
    push @files, ( map { "$dir/$_" } grep(! /^\.\.?$/, readdir DIR) ) ;
}

for (@files) {
    unless (-f $_) {
        warn "skipping $_, not a file.\n";
        next;
    }
    next unless -M $_ >= 2;
    unlink $_ or die "unlink $_: $!\n";
}
```

It would be run with a command invocation like:

```
perl expire_sessions.pl /home/you/catalogs/simple/session
```

Multiple directory names are acceptable, if there is more than one catalog.

This script can be adjusted as necessary. Refinements might include reading the file to "eval" the session reference and expire only customers who are not members.

2.5. My session files change to owner root every day!

You have the expireall `-r` entry in the root crontab, and it should either be in the Interchange user crontab or run as:

```
44 4 * * * su -c "/INTERCHANGE_ROOT/bin/expireall -r" INTERCHANGE_USERNAME
```

3. Interchange Components

Interchange components are merely portions of HTML/ITL that are included into pages within the site depending on options set in the Admin UI. The default component set includes the following:

```
best_horizontal
best_vertical
cart
cart_display
cart_tiny
category_vertical
cross_horizontal
cross_vertical
promo_horizontal
promo_vertical
random_horizontal
random_vertical
upsell_horizontal
upsell_vertical
```

3.1. Content −> Page Edit

The Interchange Admin UI offers a page editor function that allows component definitions and options to be modified for each page within the catalog.

3.1.1. Template

The choices for the Template drop−down are read from template definition files located in the CATROOT/template directory. These files store the name and description of the template, as well as components and options for the particular template.

To create a new template for use in the admin, it is best to copy an existing template definition to a new file name and edit it's contents to suit. Once the catalog is reconfigured, the new choice will be visible within the Content Page Editor admin function.

Each template option is looped through and a scratch is set using its same name and value.

ITL is used near the bottom of this file to set each option to default values before the page is displayed.

```
[set page_title][set]
[set page_banner][set]
[set members_only][set]
[set component_before][set]
[set component_after][set]
[set bgcolor]#FFFFFF[/set]
```

3.1.2. Page Title

Sets the title of the page which is synonymous with the html <title></title> code.

The following code within the template definition file is used to display this option within in the content editor:

page_title: description: Page title

This code dynamically adds the title to the page:

<title>[scratch page_title]</title>

3.1.3. Page Banner

Sets a textual title for each page which is called by [either][scratch page_banner][or][scratch page_title][/either] This results in the Page Banner being displayed if defined. Otherwise, the Page Title is used.

3.1.4. Members Only

The members only function is handled by the following code within each template file:

```
[if scratch members_only]
    [set members_only][/set]
    [if !session logged_in]
    [set mv_successpage]@@MV_PAGE@@[/set]
    [bounce page=login]
    [/if]
[/if]
```

This code says if members only is set to yes and the visitor is logged in, display the page. Otherwise, redirect the visitor to the login page.

3.1.5. Break 1

This code causes a separation in the Content Editor between the next set of options. (A blue line)

3.1.6. Horizontal Before Component

This allows the inclusion of a defined component to be displayed before, or above, the page's content. It is called with the following code within the LEFTRIGHT_TOP template:

```
[if scratch component_before]
[include file="templates/components/[scratch component_before]"]
[set component_before][/set]
[/if]
```

3.1.7. Horizontal After Component

This function allows the inclusion of a defined component to be displayed after or below the pages's content. It's called with the following code within the LEFTRIGHT_BOTTOM template:

```
[if scratch component_after]
[include file="templates/components/[scratch component_after]"]
[set component_after][/set]
[/if]
```

3.1.8. Horizontal Item Width

This setting allows you to choose how many items the horizontal components display. For example, the horizontal best sellers component uses this setting to randomly select the best sellers. Notice the default to 2 if nothing is defined.

```
random="[either][scratch component_hsize][or]2[/either]"
```

3.1.9. Special Tag

This setting is only viable when a promotion is used for a horizontal component. It tells the promotional component which rows to evaluate in the merchandising table for display within the component. This setting normally corelates to the featured column of the merchandising table as follows:

```
[query arrayref=main
          sql="
               SELECT sku,timed_promotion,start_date,finish_date
               FROM merchandising
               WHERE featured = '[scratch hpromo_type]'
               "][/query]
```

3.1.10. Before/After Banner

Allows a title for the horizontal components to be defined to displayed in a header above the component's items. It is called with the [scratch hbanner] tag.

3.1.11. Break 2

This code causes a separation in the Content Editor between the next set of options. (A blue line)

3.1.12. Vertical Component

Defines a component to be displayed along the right side of the LEFTRIGHT_BOTTOM template. It is called from the template with the following code:

```
[include file="templates/components/[scratch component_right]"]
```

3.1.13. Vertical Items Height

Sets the number of items to display within the vertical component. Called with the following code:

```
random="[either][scratch component_vsize][or]3[/either]"
```

3.1.14. Right Banner

Allows a title to be set for a vertical component which is displayed as a header above the items in the vertical component. It's called with the [scratch vbanner] tag.

3.1.15. Special Tag

Essentially the same as the Special Tag setting described in item number 9 above.

3.1.16. Background Color

Allows the background color of the page to be selected. The choices are stored within the page or template definition as in:

```
bgcolor:
        options: #FFFFFF=White,pink=Pink
        widget: select
        description: Background color
```

3.1.17. Content

Allows the page code to be downloaded, uploaded and viewed/edited. Only the code between <!−− BEGIN CONTENT −−> and <!−− END CONTENT −−> is displayed or can be edited here.

3.1.18. Preview, Save, and Cancel buttons

Allows the changes to the edited page to be previewed in a pop−up browser window, or saved. Cancel returns you to the page editor selection page.

3.1.19. Save template in page

Template settings are stored in the template definitions by default. This allows a common set of choices for template settings for all pages. If specific setting options are desired for a page, the template can be saved within the page so that it may have individual options.

The in−page template definition must be surrounded by [comment] [/comment].

3.2. Custom Admin UI Options

Other options may be added to the template by defining them in the default definition file, or using in−page definitions.

When the following lines are added to the template definition, the new option is added to the Admin UI.

```
option_name:
options: 1,2*,3
widget: select
description: Option Description
help: Other Details
```

Each time the template is used, an option_name scratch variable is created. (Called with: [scratch option_name].) This scratch value will be equal to what's selected here in the admin tool.

The possible widgets include:

```
break − produces the blue line separator.
```

```
radio − produces radio button type selections.
select − standard drop−down selector.
move_combo − select drop down with options and text input for new option.
```

4. Administrative Pages

With Interchange's `GlobalSub` capability, very complex add−on schemes can be implemented with Perl subroutines. And with the new writable database, pages that modify the catalog data can be made. See `MasterHost`, `RemoteUser`, and `Password`.

In addition, any Interchange page subdirectory can be protected from access by anyone except the administrator if a file called '.access' is present and non−zero in size.

4.1. Controlling Access to Certain Pages

If the directory containing the page has a file `.access` and that file's size is zero bytes, access can be gated in one of several ways.

1. If the file `.access_gate` is present, it will be read and scanned for page−based access. The file has the form:

```
page: condition
*: condition
```

The `page` is the file name of the file to be controlled (the .html extension is optional). The `condition` is either a literal `Yes/No` or Interchange tags which would produce a `Yes` or `No` (1/0 work just fine, as do true/false).
The entry for `*` sets the default action if the page name is not found. If pages will be allowed by default, set it to `1` or `Yes`. If pages are to be denied by default in this directory, leave blank or set to `No`. Here is an example, for the directory `controlled`, having the following files:

```
−rw−rw−r−−   1 mike     mike            0 Jan  8 14:19 .access
−rw−rw−r−−   1 mike     mike          185 Jan  8 16:00 .access_gate
−rw−rw−r−−   1 mike     mike          121 Jan  8 14:59 any.html
−rw−rw−r−−   1 mike     mike          104 Jan  8 14:19 bar.html
−rw−rw−r−−   1 mike     mike          104 Jan  8 14:19 baz.html
−rw−rw−r−−   1 mike     mike          104 Jan  8 14:19 foo.html
```

The contents of `.access_gate`:

```
foo.html: [if session username eq 'flycat']
            Yes
          [/if]
bar:      [if session username eq 'flycat']
          [or scratch allow_bar]
            Yes
          [/if]
baz:      yes
*:        [data session logged_in]
```

The page `controlled/foo` is only allowed for the logged−in user **flycat**.
The page `controlled/bar` is allowed for the logged−in user **flycat**, or if the scratch variable `allow_bar` is set to a non−blank, non−zero value.
The page `controlled/baz` is always allowed for display.
The page `controlled/any` (or any other page in the directory not named in `.access_gate`) will be allowed for any user logged in via *UserDB*. NOTE: The `.access_gate` scheme is available for database

access checking if the database is defined as an `AdminDatabase`. The `.access_gate` file is located in `ProductDir`.

1. If the Variable `MV_USERDB_REMOTE_USER` is set (non−zero and non−blank), any user logged in via the UserDB feature will receive access to all pages in the directory. NOTE: If there is a `.access_gate` file, it overrides this.
2. If the variables `MV_USERDB_ACL_TABLE` is set to a valid database identifier, the UserDB module can control access with simple ACL logic. See USER DATABASE. NOTE: If there is a `.access_gate` file, it overrides this. Also, if `MV_USERDB_REMOTE_USER` is set, this capability is not available.

4.2. display tag and mv_metadata

Interchange can store meta information for selected columns of tables in a site's database. This meta information is used when the user interacts with the database. For example, the meta informaton for a `Hide Item` field might specify that a checkbox be displayed when the user edits that field, since the only reasonable values are `on` and `off`. Or, the meta information might specify a filter on data entered for a `Filename` field which makes sure that the characters entered are safe for use in a filename.

`Widget type` specifies the `HTML INPUT` tag type to use when displaying the field in, say, the item editor.

`Width` and `Height` only apply to some of the `Widget type` options, for instance the `Textarea` widget.

`Label` is displayed instead of the internal column name. For example, the `category` column of the `products` table might have a label of `Product Category`.

`Help` is displayed below the column label, and helps describe the purpose of the field to the user.

`Help url` can be used to link to a page giving more information on the field.

`Lookup` can be used when a field is acting like a foreign key into another table. In that case, use some sort of select box as the widget type, and if referencing multiple rows in the destination table, use a multi select box, with `colons_to_null` as the `pre_filter`, and `::` as the `lookup_exclude`.

`Filter` and `pre_filter` can be used to filter data destined for that field or data read from that field, respectively.

Repeat?: The Interchange back office UI uses the mv_metadata table to discover formatting information for field, table, and view display. The information is kept in fields in the mv_metadata table, and is used to select the display, the HTML input type, the size (height and width where appropriate), label, help text, additional help URL, and default value display.

It works in conjunction with the [display ...] usertag defined in the Interchange UI as well as in specific pages in the UI. The [display] tag has this syntax:

[display table=tablename column=fieldname key=key arbitrary=tag filter=op ...]

In the simplest use, the formatting information for a table form field is called with:

[display table=products column=category key="os28007"]

The mv_metadata table is scanned for the following keys:

products::category::os28007
products::category

If a row is found with one of those keys, then the information in the row is used to set the display widget. If no row is found, an INPUT TYPE=TEXT widget is displayed. If the data is all digits, a size of 8 is used, otherwise the size is 60.

If the following row were found (not all fields shown, would be tab−separated in the actual data):

```
code                type   width height label     options
products::category  text   20           Category
```

Then this would be output:

```
<INPUT TYPE=text SIZE=20 VALUE="*category*">
```

If the following row were found:

```
code                type   width height label     options
products::category  select              Category  =none, product=Hardware
```

Then the following would be output:

```
<SELECT NAME=category>
<OPTION VALUE="">none
<OPTION VALUE="product">Hardware
</SELECT>
```

The standard widget types are:

text

The default. Uses the fields:

```
width   size of input box
```

textarea

Format a <TEXTAREA> </TEXTAREA> pair. Uses the fields:

```
width   COLS for textarea
height  ROWS for textarea
```

select

Format a <SELECT> </SELECT> pair with appropriate options. Uses the fields:

```
height          SIZE for select
default         Default for SELECTED
options         Options for a fixed list (or prepended to a lookup)
lookup          signals a lookup (used as field name if "field" empty)
field           field to look up possible values in
```

```
db              table to look up in if not current table
lookup_exclude  regular expression to exclude certain values from lookup
```

5. Usertag Reference

Admin Tool−specific usertags.

6. Admin Tool Database Tables

6.1. icmenu.txt

Used for back−office administration UI menus and wizards.

```
code
            Arbitrary primary key
mgroup
            Menu group (for grouping searches)
msort
            Sort order within the group
next_line
            Set to 1 if submenu
indicator
exclude_on
depends_on
page
form
name
super
inactive
special
help_name
    img_dn
    img_up
    img_sel
    img_icon
    url
```

6.2. mv_metadata.asc

```
code
    Table::Column to be operated on.
    Database table
type
    Widget type (Select the basic display type for the field)
        textarea = Textarea
        text = Text Entry (default)
        select = Select Box
        yesno = Yes/No (Yes=1)
        noyes = No/Yes (No=1)
        multiple = Multiple Select
        combo = Combo Select
        reverse_combo = Reverse Combo
        move_combo = Combo move
        display = Text of option
        hidden_text = Hidden(show text)
        radio = Radio box
        radio_nbsp = Radio (nbsp)
        checkbox = Checkbox
        check_nbsp = Checkbox (nbsp)
        imagedir = Image listing
        imagehelper = Image upload
        date = Date selector
        value = Value
        option_format = Option formatter
        show = Show all options
```

```
width
    Width (SIZE for TEXT, COLS for TEXTAREA, Label limit for SELECT)
height
    Height (SIZE for SELECT, ROWS for TEXTAREA)
field
    Field for lookup (can be two comma separated fields, in which case
    second is used as the label text.  Both must be in the same table.)
db
name
    Variable name (normally left empty, changes variable name to send in
    form)
outboard
    Select directory for image listing widget
options
    options in the format <blockquote>value=label*</blockquote>
attribute
    Column name (Do not set this.)
label
help
    Help (displays at top of page)
lookup
    Lookup select (Whether lookup is performed to get options for a select
    type.  If nothing is in the field, then used as the name of the field
    to lookup in.  Use lookup table if you want to look up in a different
    table.
filter
    Filters (Filters which can transform or constrain your data.  Some
    widgets require filters.)
help_url
    Help URL (links below help text)
    A URL which will provide more help
pre_filter
lookup_exclude
    ADVANCED: regular expression that excludes certain keys from the lookup
prepend
append
    Append HTML (HTML to be appended to the widget.  Will substitute in the
    macros _UI_TABLE_, _UI_COLUMN_, _UI_KEY_, and _UI_VALUE_, and will
    resolve relative links with absolute links.)
display_filter
```

7. makecat − Set Up a Catalog from a Template

After Interchange is installed, you need to set up at least one catalog. Interchange will not function properly until a catalog is created.

The supplied `makecat` script, which is in the Interchange program directory `bin`, is designed to set up a catalog based on the user's server configuration. It interrogates the user for parameters like which directories to use, a URL to base the catalog in, HTTP server definitions, and file ownership. It gives relevant examples of the entries it expects to receive.

Note: A catalog can only be created once. All further configuration is done by editing the files within the catalog directory.

The `makecat` script requires a catalog skeleton to work from. The Foundation demo is distributed with Interchange. See the icfoundation document for information on building the Foundation demo store. Other demo catalogs are available at `http://interchange.redhat.com/`.

It is not normally necessary for you to understand how to build catalog skeletons for use with makecat, but the following information will help you if you should ever need to.

7.1. Catalog Skeletons

A catalog skeleton contains an image of a configured catalog. The best way to see what the makecat program does is to configure the simple demo and then run a recursive `diff` on the template and configured catalog directories:

```
cd /usr/local/interchange
diff −r construct catalogs/construct
```

The files are mostly identical, except that certain macro strings have been replaced with the answers given to the script. For example, if `www.mydomain.com` was answered at the prompt for a server name, this difference would appear in the catalog.cfg file:

```
# template
Variable SERVER_NAME  __MVC_SERVERNAME__

# configured catalog
Variable SERVER_NAME  www.mydomain.com
```

The macro string __MVC_SERVERNAME__ was substituted with the answer to the question about server name. In the same way, other variables are substituted, and include:

```
MVC_BASEDIR      MVC_IMAGEDIR
MVC_CATROOT      MVC_IMAGEURL
MVC_CATUSER      MVC_MAILORDERTO
MVC_CGIBASE      MVC_MINIVENDGROUP
MVC_CGIDIR       MVC_MINIVENDUSER
MVC_CGIURL       MVC_SAMPLEHTML
MVC_DEMOTYPE     MVC_SAMPLEURL
MVC_DOCUMENTROOT MVC_VENDROOT
MVC_ENCRYPTOR
```

Note: Not all of these variables are present in the "construct" template, and more may be defined. In fact, any environment variable that is set and begins with MVC_ will be substituted for by the `makecat` script. For example, to set up a configurable parameter to customize the COMPANY variable in catalog.cfg, run a pre−qualifying script that set the environment variable MVC_COMPANY and then place in the catalog.cfg file:

Variable COMPANY __MVC_COMPANY__

All files within a template directory are substituted for macros, not just the catalog.cfg file. There are two special directories named `html` and `images`. These will be recursively copied to the directories defined as SampleHTML and ImageDir.

Note: The template directory is located in the Interchange software directory, i.e., where `interchange.cfg` resides. Avoid editing files in the template directory. To create a new template, it is recommended that it should be named something besides 'construct' and a copy of the `construct` demo directory be used as a starting point. Templates are normally placed in the Interchange base directory, but can be located anywhere. The script will prompt for the location if it cannot find a template.

In addition to the standard parameters prompted for by Interchange, and the standard catalog creation procedure, four other files in the `config` directory of the template may be defined:

```
additional_fields −− file with more parameters for macro substitution
additional_help   −− extended description for the additional_fields
precopy_commands  −− commands passed to the system prior to catalog copy
postcopy_commands −− commands passed to the system after catalog copy
```

All files are paragraph−based. In other words, a blank line (with no spaces) terminates the individual setting.

The additional_fields file contains:

```
PARAM
The prompt. Set PARAM to?
The default value of PARAM
```

This would cause a question during makecat:

```
The prompt. Set PARAM to?.....[The default value of PARAM]
```

If the additional_help file is present, additional instructions for PARAM may be provided.

```
PARAM

These are additional instructions for PARAM, and they
may span multiple lines up to the first blank line.
```

The prompt would now be:

```
These are additional instructions for PARAM, and they
may span multiple lines up to the first blank line.

The prompt. Set PARAM to?.....[The default value of PARAM]
```

If the file config/precopy_commands exists, it will be read as a command followed by the prompt/help value.

```
mysqladmin create __MVC_CATALOGNAME__
We need to create an SQL database for your Interchange
database tables.
```

This will cause the prompt:

```
We need to create an SQL database for your Interchange
database tables.

Run command "mysqladmin create simple"?
```

If the response is "y" or "yes," the command will be run by passing it through the Perl system() function. As with any of the additional configuration files, MVC_PARAM macro substitution is performed on the command and help. Proper permissions for the command are required.

The file config/postcopy_commands is exactly the same as <precopy_commands>, except the prompt occurs after the catalog files are copied and macro substitution is performed on all files.

There may also be SubCatalog directives:

```
SubCatalog easy simple /home/catalogs/simple /cgi-bin/easy
```

easy

The name of the subcatalog, which also controls the name of the subcatalog configuration file. In this case, it is `easy.cfg`.

simple

The name of the base configuration that will be the basis for the catalog. Parameters in the easy.cfg file that are different will override those in the catalog.cfg file for the base configuration.

The remaining parameters are similar to the Catalog directive.

Additional interchange.cfg parameters set up administrative parameters that are catalog wide. See the server configuration file for details on each of these.

Each catalog can be completely independent with different databases, or catalogs can share pages, databases, and session files. This means that several catalogs can share the same information, allowing "virtual malls."

7.2. Manual Installation of Catalogs

An Interchange installation is complex, and requires quite a few distinct steps. Normally you will want to use the interactive catalog builder, `makecat`, described above. It makes the process much easier. Please see the `iccattut` document for a full tutorial on building a catalog by hand.

8. Link Programs

Interchange requires a web server that is already installed on a system. It does have an internal server which can be used for administration, testing, and maintenance, but this will not be useful or desireable in a production environment.

As detailed previously, Interchange is always running in the background as a daemon, or resident program. It monitors either a UNIX−domain file−based socket or a series of INET−domain sockets. The small CGI link program, called in the demo `simple`, is run to connect to one of those sockets and provide the link to a browser.

Note: Since Apache is the most popular web server, these instructions will focus on it. If using another type of web server, some translation of terms may be necessary.

A `ScriptAlias` or other CGI execution capability is needed to use the link program. (The default `ScriptAlias` for many web servers is `/cgi−bin`.) If `ExecCGI` is set for all directories, then any program ending in a particular file suffix (usually `.cgi`) will be seen as a CGI program.

Interchange, by convention, names the link program the same name as the catalog ID, though this is not required. In the distribution demo, this would yield a program name or SCRIPT_PATH of `/cgi−bin/simple` or `/simple.cgi`. This SCRIPT_PATH can be used to determine which Interchange catalog will be used when the link program is accessed.

8.1. UNIX−Domain Sockets

This is a socket which is not reachable from the Internet directly, but which must come from a request on the server. The link program `vlink` is the provided facility for such communication with Interchange. This is the most secure way to run a catalog, for there is no way for systems on the Internet to interact with Interchange except through its link program.

The most important issue with UNIX−domain sockets on Interchange is the permissions with which the CGI program and the Interchange server run. To improve security, Interchange normally runs with the socket file having 0600 permissions (rw−−−−−−−), which mandates that the CGI program and the server run as the same user ID. This means that the `vlink` program must be SUID to the same user ID as the server executes under. (Or that CGIWRAP is used on a single catalog system).

With Interchange's multiple catalog capability, the permissions situation gets a bit tricky. Interchange comes with a program, `makecat`, which configures catalogs for a multiple catalog system. It should properly set up ownership and permissions for multiple users if run as the superuser.

8.2. INET−Domain Sockets

These are sockets which are reachable from the Internet directly. The link program `tlink` is the provided facility for such communication with Interchange. Other browsers can talk to the socket directly if mapped to a catalog with the global `TcpMap` directive. To improve security, Interchange usually checks that the request comes from one of a limited number of systems, defined in the global `TcpHost` directive. (This check is not made for the internal HTTP server.)

8.3. Internal HTTP Server

If the socket is contacted directly (only for INET−domain sockets), Interchange will perform the HTTP server function itself, talking directly to the browser. It can monitor any number of ports and map them to a particular catalog. By default, it only maps the special catalog mv_admin, which performs administrative functions. The default port is 7786, which is the default compiled into the distribution tlink program. This port can be changed via the `TcpMap` directive.

To prevent catalogs that do not wish access to be made in this way from being served from the internal server, Interchange has a fixed SCRIPT_PATH of `/catalogname` (/simple for the distribution demo) which needs to be placed as an alias in the Catalog directive to enable access. See `TcpMap` for more details.

8.4. Setting Up VLINK and TLINK

The `vlink` and `tlink` programs, compiled from `vlink.c` and `tlink.c`, are small C programs which contact and interface to a running Interchange daemon. The VLINK executable is normally made setuid to the user account which runs Interchange, so that the UNIX−domain socket file can be set to secure permissions (user read−write only). It is normally not necessary for the user to do anything. They will be compiled by the configuration program. If the Interchange daemon is not running, either of the programs will display a message indicating that the server is not available. The following defines in the produced `config.h` should be set:

LINK_FILE

Set this to the name of the socket file that will be used for configuration, usually "/usr/local/lib/interchange/etc/socket" or the "etc/socket" under the directory chosen for the VendRoot.

LINK_HOST

Set this to the IP number of the host which should be contacted. The default of 127.0.0.1 (the local machine) is probably best for many installations.

LINK_PORT

Set this to the TCP port number that the Interchange server will monitor. The default is 7786 (the decimal ASCII codes for 'M' and 'V') and does not normally need to be changed.

LINK_TIMEOUT

Set this to the number of seconds `vlink` or `tlink` should wait before announcing that the Interchange server is not running. The default of 45 is probably a reasonable value.

8.5. Compiling VLINK and TLINK

There is a `compile_link` program which will assist with this. Do:

```
perldoc VENDROOT/bin/compile_link
```

for its documentation.

8.6. Manually Compiling VLINK and TLINK

Change directories to the `src` directory, then run the GNU configure script:

```
cd src
./configure
```

There will be some output displayed as the configure script checks the system. Then, compile the programs:

```
perl compile.pl
```

To compile manually:

```
cc vlink.c −o vlink
cc tlink.c −o tlink
```

On manual compiles, ensure that the C compiler will be invoked properly with this little ditty:

```
perl −e 'do "syscfg"; system("$CC $LIBS $CFLAGS $DEFS −o tlink tlink.c");'
perl −e 'do "syscfg"; system("$CC $LIBS $CFLAGS $DEFS −o vlink vlink.c");'
```

On some systems, the executable can be made smaller with the strip program, if available. It is not required.

```
strip vlink
strip tlink
```

If Interchange is to run under a different user account than the individual configuring the program, make that user the owner of `vlink`. Do not make `vlink` owned by root, because making `vlink` SETUID root is an huge and unnecessary security risk. It should also not normally run as the default Web user (often `nobody` or `http`)).

```
chown interchange vlink
```

Move the `vlink` executable to the cgi−bin directory:

```
mv vlink /the/cgi−bin/directory
```

Make `vlink` SETUID:

```
chmod u+s /the/cgi−bin/directory/vlink
```

Most systems unset the SUID bit when moving the file, so change it after moving.

The SCRIPT_NAME, as produced by the HTTP server, must match the name of the program. (As usual, let the makecat program do the work.)

8.7. VLINK or TLINK Compile Problems

The latest version of `vlink.c` and `tlink.c` have been compiled on the following systems:

```
AIX 4.1
BSD2.0 (Pentium/x86)
```

```
Debian GNU/Linux
Digital Unix (OSF/Alpha)
FreeBSD 2.x, 3.x, 4.x
IRIX 5.3, IRIX 6.1
OpenBSD 2.7
Red Hat Linux 6.2, 7.0, 7.1
SCO OpenServer 5.x
Solaris 2.x (Sun compiler and GCC)
Solaris 7 (Sun compiler and GCC)
SunOS 4.1.4
```

Some problems may occur. In general, ignore warnings about pointers.

Make sure that you have run the configure program in the src directory. If you use Interchange's makecat program, it will try to compile an appropriate link at that time, and will substitute tlink.pl if that doesn't work.

You can compile manually with the proper settings with this series of commands:

```
cd src
./configure
perl −e 'do "syscfg"; system ("$CC $CFLAGS $DEFS $LIBS −o tlink tlink.c")'
perl −e 'do "syscfg"; system ("$CC $CFLAGS $DEFS $LIBS −o vlink vlink.c")'
```

There is also a `compile_link` program which has docmentation embedded and which will compile an approprate link. If you cannot compile, try using the `tlink.pl` script, written in Perl instead of C, which should work on most any system. Since vlink needs to have values set before compilation, a pre−compiled version will not work unless it has the exact values you need on your system. If you can use the defaults of 'localhost' and port 7786, you may be in luck.

9. Installing Perl Modules without Root Access

Installing Interchange without root access is no problem. However, installing Perl modules without root access is a little trickier.

You must build your makefile to work in your home dir. Something like:

```
PREFIX=~/usr/local \
INSTALLPRIVLIB=~/usr/local/lib/perl5 \
INSTALLSCRIPT=~/usr/local/bin \
INSTALLSITELIB=~/usr/local/lib/perl5/site_perl \
INSTALLBIN=~/usr/local/bin \
INSTALLMAN1DIR=~/usr/local/lib/perl5/man \
INSTALLMAN3DIR=~/usr/local/lib/perl5/man/man3
```

Put this in a file, say 'installopts', and use it for the Makefile.PL.

```
perl Makefile.PL `cat installopts`
```

Then, forget ./config. Just do:

```
make
make test
make install
```

Some of the tests may fail, but that's probably ok.

Also make sure to install Bundle::Interchange, which will need the same config data as you put into 'installopts'.

10. Installation Troubleshooting

Interchange uses the services of other complex programs, such as Perl, Web servers, and relational databases, to work. Therefore, when there is a problem, check these programs before checking Interchange. Many more basic installation problems have to do with those than with Interchange itself.

If an error message is received about not being able to find libraries, or a core dump has occurred, or a segment fault message, it is always an improperly built or configured Perl. Contact the system administrator or install a new Perl.

The `makecat` program is intended to be used to create the starting point for the catalog. If the demo does not work the first time, keep trying. If it still does not work, try running in INET mode.

Check the two error log files: `error.log` in the Interchange home directory (where interchange.cfg resides) and `error.log` in the catalog directory (where catalog.cfg resides; there can be many of these). Many problems can be diagnosed quickly if these error logs are consulted.

Check the README file, the FAQ, and mail list archive at the official Interchange web site for information:

```
http://interchange.redhat.com/
```

Double check the following items:

1. Using UNIX sockets?
   - Check that the `vlink` program is SUID, or the appropriate changes have been made in the SocketPerms directive. Unless the files are world−writable, the vlink program and the Interchange server must run as the same user ID! If running CGI−WRAP or SUEXEC, the `vlink` program must not be SUID.
   - If having trouble with the vlink program (named `construct` in the demo configuration), try re−running `makecat` and using INET mode instead. (Or copy the `tlink` INET mode link program over `vlink`). This should work unchanged for many systems.
   - If using an ISP or have a non−standard network configuration, some changes to interchange.cfg are necessary. For `tlink` to work, the proper host name(s) must be configured into the TcpHost directive in interchange.cfg. The program selects port 7786 by default (the ASCII codes for "M" and "V"). If another port is used, it must be set to the same number in both the tlink program (by running `compile_link`) and the `minivend.cfg` file. The `tlink` program does not need to be SUID.
2. Proper file permissions?
   - The Interchange server should not run as the user `nobody`! The program files can be owned by anyone, but any databases, ASCII database source files, error logs, and the directory that holds them must be writable by the proper user ID, that is the one that is executing the MiniVend program.
   - The best way to operate in multi−user, multiple catalog setups is to create a special `interch` user, then put that user in the group that contains each catalog user. If a group is defined for each individual user, this provides the best security. All associated files can be in 660 or 770 mode. There should be no problems with permissions and no problems with security.
3. Is the `vlink` program being executed on a machine that has the socket file `etc/socket` on a directly attached disk?
   - UNIX−domain sockets will not work on NFS−mounted file systems! This means that the server `minivend` and the CGI program `vlink` must be executing on the same machine.

- The `tlink` program does not have this problem, but it must have the proper host name(s) and TCP ports set in the TcpHost and TcpPort directives in `interchange.cfg`. Also, be careful of security if sensitive information, like customer credit card numbers, is being placed on a network wire. line:

Catalog−Building Tutorial

11. Purpose

The purpose of this document is to guide you through constructing a simple Interchange catalog from scratch. The demo catalog that ships with Interchange is quite complex since it highlights some of the many capabilities that Interchange offers. As a template for your own catalog, the demos can either be an intimidating place to start or too customized to relate to your business.

The simple catalog you create using this tutorial should give you a feel for the basic Interchange system. It should also be considered a stepping stone to a more complete and functional e−commerce system built with Interchange. The tutorial relies as much as possible on default settings to accentuate how Interchange works. It will use as few of Interchange's capabilities as possible, while still building a usable store. The resulting site will be simple but usable. The value of this tutorial is not in the resulting e−commerce site, but in the instruction that occurs along the way.

It is recommended that you create the files used in this tutorial yourself. You will learn more by creating the directory structure and using your favorite text editor to create files in the proper places on your own system as they are discussed.

12. Before you begin

This section explains the initial set up tasks that must be completed before you can begin building your simple e−commerce site.

12.1. Install Interchange and the demo catalog

The easiest way to get Interchange and the demo set up is through an *RPM install* on the Red Hat Linux or Linux Mandrake operating systems. You can also get Interchange by unpacking an Interchange tarball or checking out a copy of the CVS repository and doing a *manual installation*. These installations can be done either as a regular user or as root with a special Interchange user.

You must also know what type of installation you ran so you know where to place the various files created. Before proceeding, verify that Interchange is properly installed. Also, which type of installation you ran:

- RPM (Red Hat Package Manager) install
- Manual install as root
- Manual install as regular user

Note: After installation, `makecat` should be run to build your catalog. For information on installing Interchange and building your catalog using `makecat`, see the *Red Hat Interchange 4.8: Getting Started Guide*. Do not to continue with this tutorial without a working demo catalog.

Installing the demo catalog set up the Interchange global configuration file `interchange.cfg`, which resides in the Interchange software directory. Also, it compiled the link program for your specific server and placed the executable program in your cgi−bin directory. This is necessary for your catalog to run properly.

12.2. The Interchange operating system user

If Interchange was installed as a regular user, that will be the user Interchange runs as. If Interchange was installed as root or from an RPM, you need to know the name of the separate Interchange user. The Interchange daemon will not run as root, or even as the web server user (often www or httpd or nobody). If Interchange was installed from the RPM, or with the default source installation settings, the default username is `interch`. If a different user name was established, you will need to know what it is.

12.3. Important directories

In order to complete this tutorial you will need to know the location of each of the following directories and have write permissions on them:

- Interchange software directory **.RPM install:** /usr/lib/interchange **.Manual install as root:** /usr/local/interchange **.Manual install as regular user:** /home/username/interchange
- Catalogs directory **.RPM install:** /var/lib/interchange **.Manual install as root:** /usr/local/interchange/catalogs **.Manual install as regular user:** /home/username/catalogs
- cgi−bin directory **.RPM install or source install as root (Red Hat 6, Linux Mandrake):** /home/httpd/cgi−bin **.RPM install or source install as root (Red Hat 7):** /var/www/cgi−bin **.Manual install as root (locally installed web server):** /usr/local/htdocs, /opt/www, ... **.Manual install as regular user:** /home/username/public_html (with .cgi extension)

Note: The installation of Interchange is very flexible and the file locations on your system may vary, depending on how your system was set up. It is recommended that you not proceed until you are sure you have this information and the necessary permissions to write to these directories.

12.4. Your catalog URL

Finally, you need to know the URL to access your store from a web browser. Again, this can vary depending on how your web server has been set up. But, assuming a common setup of the Apache web server, your URL should be one of the following:

- **Root or RPM install:** `http://localhost/cgi-bin/tutorial/pagename`
- **Manual install as user:** `http://localhost/~username/tutorial.cgi/pagename`

If you aren't running your web browser on the server where Interchange is running, you need to substitute your server's host name (for example: `machine.domain.com` for `localhost`) where mentioned.

Note: It is recommended that you use the real machine name instead of localhost. The standard for cookies specifies that they can only be set when a domain name has at least two dots in it. If you use localhost, you will lose session information if you leave catalog, since the session ID is passed only as part of the URL.

12.5. Starting or restarting Interchange

When you make changes to the configuration files you need to restart the Interchange server. How this is done depends on how you installed Interchange:

- **RPM install as root:** `/usr/sbin/interchange -r`
- **Manual install as Interchange user:** `/usr/local/interchange/bin/interchange -r`
- **Manual install as root:** `su interch -c '/usr/local/interchange/bin/interchange -r'`
- **Manual install as regular user:** `~/interchange/bin/interchange -r`

Find the right command for your system and remember it, since you will need to restart Interchange a few times during the tutorial.

12.6. Tutorial assumptions

Because it is impossible to cover all senarios, this tutorial assumes that you installed Interchange on Red Hat 7 from the RPM. This creates the following settings:

- **Interchange software directory:** /usr/lib/interchange
- **Catalogs directory:** /var/lib/interchange
- **cgi-bin directory:** /var/www/cgi-bin
- **Interchange user:** interch
- **Demo catalog name:** foundation
- **Demo catalog URL base:** http://localhost/cgi-bin/foundation
- **Tutorial catalog name:** tutorial
- **Tutorial catalog URL base:** http://localhost/cgi-bin/tutorial

- **Tutorial catalog directory:** /var/lib/interchange/tutorial

If you did not install with these settings, substitute the correct values for your system when these settings are mentioned in the tutorial.

13. Building Your Catalog

This section describes the pages and directories that need to be established to create a properly functioning catalog.

13.1. Create the link program

You need to make a copy of the demo link program in your cgi−bin directory and name it `tutorial`.

The demo link program has the same name as your demo catalog, usually `foundation`. The link program links the Interchange daemon with your web server. Make sure that it has the same owner and file permissions as the one you copied from. The set−UID bit is especially (unless you installed as a regular user). Normally you will need to be root to have write permissions in the cgi−bin directory.

Type this command as root while in your cgi−bin directory:

```
cp −p foundation tutorial
```

If everything is working correctly, typing `ls −l` should describe your files roughly like this:

```
−rwsr−xr−x    1 interch  interch      7708 Dec 16 22:47 foundation
−rwsr−xr−x    1 interch  interch      7708 Dec 16 22:47 tutorial
```

13.2. Create the tutorial catalog directory

As root, create a subdirectory named `tutorial` under your catalogs directory (probably `/var/lib/interchange/`). This is where all of the catalog−specific files will go. It needs to be readable, writable, and executable by the Interchange user. This will be referred to as your catalog directory. Type the following while in the catalogs directory to create the tutorial subdirectory:

```
mkdir tutorial
chown interch.interch tutorial
chmod 770 tutorial
```

13.3. Become the Interchange user

You should be able to do everything you need to do as the 'interch' user for the rest of this tutorial. So you can switch to that user now (`su interch`). If you installed Interchange from the RPM, the user **interch** probably doesn't have a password. You'll have to set it with a command such as `passwd interch` while root.

13.4. Go to the tutorial catalog directory

Change to the catalog directory with the 'cd' command. For the rest of this tutorial, all file locations will be given relative to the tutorial catalog directory. For example, `pages/ord/basket.html` would actually be `/var/lib/interchange/tutorial/pages/ord/basket.html` or the equivalent on your system. The only exception is `interchange.cfg`, which is in the Interchange software directory.

Note: To improve clarity, we will append a trailing slash to directory names to clearly distinguish them from file names. (Similar to the output of the `ls` command with the `-F` option.)

13.5. Create the session directory

You need to create the session directory where Interchange saves information on each visitor's browsing session. If you do not have this directory, Interchange may fail to work. This directory is called `session/` and goes under your catalog directory. Type `mkdir session` to create this directory.

14. Configuration files

Interchange configuration is controlled by a number of directives, which are specified in two files. Global configuration directives go in `interchange.cfg` in the Interchange software directory. Catalog−specific configuration directives go in `catalog.cfg` in the catalog directory.

A complete directive consists of the directive name followed by whitespace−separated parameters. Any number of spaces or tabs can be between the directive and its options, but the directive and its options must be on the same line. The directive is case−insensitive, but it is recommended that you use it consistently for readability.

You can insert blank lines or comment lines (lines where the first non−blank character is '#') throughout the configuration files to improve readability. The order the lines appear in is significant, but unimportant for the simple catalog you are creating.

For the next part, access your text editor (for example, vi, emacs, pico, joe, gedit, or nedit) to start editing some files.

14.1. interchange.cfg

The first directive we need to use is a global directive that tells Interchange where the new catalog is, called *Catalog*. The ***Catalog*** directive has the following format:

```
Catalog   name   catalog_base_directory   link_url_path
```

Open `interchange.cfg` in the Interchange software directory. Go near the top of the file, right below the other Catalog directives, and add this line:

```
Catalog  tutorial  /var/lib/interchange/tutorial  /cgi−bin/tutorial
```

Save the file.

14.2. catalog.cfg

For the rest of the tutorial, most of the files mentioned do not exist yet. You will create them yourself with initial text we give.

You need to create a `catalog.cfg` file for your tutorial store (in the tutorial catalog directory). We'll start with a very simple products database table with a few fields and a few products.

The ***Database*** directive describes a database table to the Interchange system in this format:

```
Database  name  filename  format
```

Interchange has several database options available. We will use the simplest, which is the built−in default (specifically, some variant of DBM). The default location for *filename* is in a subdirectory called `products` under the catalog directory. Interchange recongnizes a number of file formats. We will use a tab−delimited text file. Enter the following into `catalog.cfg`:

```
Database  products  products.txt  TAB
```

This tells Interchange that you have a database table named 'products' that is described in a tab−delimited file named `products.txt`. You can describe an unlimited number of arbitrary database tables for the system to use this way. Interchange is an e−commerce system and it expects at least a products database table. You can specify all of the database tables that contain products by using the ***ProductFiles*** directive. There is no default for this, so you will have to specify your products database by adding the following line to `catalog.cfg`:

```
ProductFiles  products
```

There are a few other directives that Interchange expects to see in order to complete the minimum configuration. They are ***VendURL***, ***SecureURL***, and ***MailOrderTo***. They are, respectively, your catalog's base URL, its secure URL, and the e−mail address to mail order notices to. Add the following lines to `catalog.cfg` to establish these directives:

```
VendURL  http://localhost/cgi-bin/tutorial
SecureURL  http://localhost/cgi-bin/tutorial
MailOrderTo  your@email.address
```

The `catalog.cfg` file should look like this when you save it:

```
Database  products  products.txt  TAB
ProductFiles  products
VendURL  http://localhost/cgi-bin/tutorial
SecureURL  http://localhost/cgi-bin/tutorial
MailOrderTo  your@email.address
```

15. The products database table

15.1. products/products.txt

Create the `products/` directory in your tutorial catalog directory.

The `products/products.txt` file will serve two purposes. It will provide Interchange with the layout of the products database table and it will also provide the data. When Interchange parses the products.txt file, it will expect the first line to contain the names of the fields for the database table (for example, sku, description, price). The first field in the list is expected to be a primary key (unique identifier) for that row. In most cases you are going to use the SKU (stock keeping unit) as the unique identifier for each product.

The product database is handled as a special case since Interchange expects at least the description, price, and product ID (sku) fields. In other words, the `products.txt` file must at least contain fields named sku, price, and description. Any other fields you decide to include are handled normally.

The simple store that we are going to build will sell tests. You can choose another sample product line, but it is recommended that you keep it simple. Create the file `products/products.txt` to look like this, with a single tab separating each field:

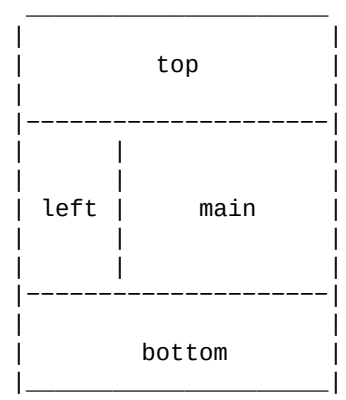
```
sku   description     price
4595  Nice Bio Test   275.45
2623  Stack of Econ Quizzes   1.24
0198  Really Hard Physics Test        1589.34
1299  Ubiquitous diff eq final        37.00
```

Note: When using tab−delimited files as we are, make sure you have exactly one tab between each field. Some text editors will use spaces to simulate tabs. Interchange expects actual ASCII tab characters; no spaces or extra characters are accepted.

You may notice that the columns don't line up in your text editor. This is the nature of tab−delimited files. Do not try to fix these.

16. Page templates

Since most sites have certain aspects of the site that remain the same as the content of the pages changes, we are going to create a template that we can use for all pages. We'll divide the page into four sections:



The "main" section holds the content that is different for each page. The "top" section is for headers, banners, menus, and so on. The "left" section can be used as a sidebar or navagation bar, and the "bottom" section can contain the copyright and contact info. The top, left, and bottom sections will remain constant throughout the site. Making a change to information in one of these sections will make that change to all pages in your site.

Now type the HTML for each template section in an individual plain text file in the catalog directory, named 'top', 'left', and 'bottom', respectively using the code displayed below. No '.html' suffixes are used on these because they are not meant to be parsed directly by Interchange as full pages.

16.1. top

```
<html>
<head>
<title>The Interchange Test Catalog</title>
</head>
<body>
<div align=center>
<table width="80%" border cellpadding=15>
<tr><td colspan=2 align=center><h1>The Interchange Test Catalog</h1></td></tr>
```

16.2. left

```
<tr>
<td align=center>(left)</td>
<td align=center>
```

16.3. bottom

```
</td>
</tr>
<tr><td colspan=2 align=center>(bottom)</td></tr>
</table>
</div>
</body>
```

```
</html>
```

16.4. The Interchange Tag Language

Now we need a way to pull the template pieces we just created into the proper places to make a complete page. This is done using ITL, the Interchange Tag Language.

ITL is at the heart of almost all Interchange catalog pages. It's how you use Interchange's functionality. The ITL tags appear between square brackets like [this]. Options appear after the tag, separated by whitespace, like this: [tag option1 option2] and this: [tag option1=value1 option2=value2]. They can span multiple lines. (That can help readability when the tag has many options.) There are many ITL tags, and for this tutorial very few will be addressed. For a complete listing of the ITL tags, see the *Interchange Tag Reference Guide*.

Your first tag will be [include], which reads the file mentioned (relative to the catalog directory), parses any Interchange tags, and puts the result in place of the tag. This is demonstrated on the next page you need to create.

17. Creating a welcome page

17.1. pages/index.html

Create a directory called `pages/` in your tutorial catalog directory.

Type the following text and save it as `pages/index.html`. This will create a page to test that everything works so far.

```
[include top]
[include left]
This is where your content goes.
[include bottom]
```

Restart Interchange so your changes take effect. Go to your web browser and load the page. The URL should be similar to the following: `http://localhost/cgi-bin/tutorial/index.html`.

Note: Interchange pages in the `pages/` or other directories **must** have the .html suffix on them. You can drop the suffix in your URL and in other places, such as the [page] tag you'll learn about later, but the file name itself must have the suffix.

---

18. Troubleshooting

Your first Interchange page should have displayed as described in your browser. If it didn't, you need to figure out what went wrong. Most of the time, overlooked details are the problem. Double−checking your typing is a good habit to get into.

The following is a troubleshooting checklist to use when you run into problems:

1. Have you created directories with the proper names in the proper locations? (See Appendix A for a full directory and file structure of the tutorial catalog.)
2. Have you misspelled any file names or put them in the wrong directories? Are the files and parent directories readable by the `interch` user? Double−check with the `ls` command.
3. Did you type letters in the proper case? Remember that both Unix and Interchange are case−sensitive, and for the most part you may not switch upper− and lower−case letters.
4. Did you type all punctuation, ITL tags, and HTML tags correctly?
5. Did you use whitespace correctly in the cases where it mattered? Remember to use tabs when tabs are called for (in lists and database text files).
6. Did you restart Interchange if you changed anything in `interchange.cfg` or `catalog.cfg`, or if you're in a high−traffic mode?
7. Check your catalog error log, `error.log` in your tutorial catalog directory, to see if Interchange reported any errors.
8. Check the Interchange server error log, `error.log` in the Interchange software directory, to see if it had problems loading the catalog at all.
9. View the HTML source of any catalog pages that are loading incorrectly to check for a coding error. The problem may reveal itself when you see what HTML the browser is getting.

19. Displaying products

19.1. Listing all products

Now that your store is running, you need to display your products on the welcome page. We will loop over all of the products in our database and produce an entry for each one in a table. Replace the line "This is where your content goes" in `pages/index.html` with the following:

```
<table cellpadding=5>
<tr>
<th>Test #</th>
<th>Description</th>
<th>Price</th>
</tr>

. . .

</table>
```

Now we will use Interchange tags to fill in the rest of the table from the products database you created. The [loop] [/loop] ITL tag pair tells Interchange to iterate over each item in the parameter list. In this case, the loop is over the result of an Interchange search. The search parameter does a database search on the provided parameters. In this case, we're doing a very simple search that returns all of the fields for all of the entries in the products database. The parameters passed to the search tell Interchange to *return all* ('ra') on the file ('fi') *products* respectively. The following should take the place of the ellipsis in the code you placed in `index.html`:

```
[loop search="ra=yes/fi=products"]

. . .

[/loop]
```

In the loop we just established, the individual elements of the entry using the [loop−field] tag. The following code should replace the above ellipsis in the code we placed in `pages/index.html`:

```
<tr>
<td>[loop−code]</td>
<td>[loop−field description]</td>
<td align=right>[loop−field price]</td>
</tr>
```

The [loop−code] tag refers to the primary key (unique identifier) for the current row of the database table in question. In this case, it will produce the same output as the [loop−field sku] tag, because the 'sku' field is the primary key for products table. In each case the tag is replaced by the appropriate element. When put together, Interchange generates a page with your products table on it.

Your finished page should look like this:

```
[include top]
[include left]
<table cellpadding=5>
<tr>
<th>Test #</th>
```

```
<th>Description</th>
<th>Price</th>
</tr>
[loop search="ra=yes/fi=products"]
<tr>
<td>[loop-code]</td>
<td>[loop-field description]</td>
<td align=right>[loop-field price]</td>
</tr>
[/loop]
</table>
[include bottom]
```

Test this page by refreshing the `index.html` page in your browser.

19.2. pages/flypage.html

The next step is to create an individual page for each item. To do this, you need to create a special generic page called `pages/flypage.html`. When a page is requested that does not exist in the `pages/` directory, Interchange will check and see if the requested page has the same name as a product ID from the product database table (in this case a SKU). If it does, it will show the flypage for that product. If there's no product with that ID, the special error page `special_pages/missing.html` (described in the next section) will be displayed.

For example, if the page `0198.html` was requested, Interchange first checks for a page with that name. If one is not found, it searches the products database table for a product with that ID. Interchange then creates a product page "on the fly" using `pages/flypage.html`. When constructing the flypage, the entire product record for the requested product is available through the [item-field] tag (similar to the [loop-field] tag). To create a fly page, type the following code and save it as `pages/flypage.html`.

```
[include top]
[include left]

<h3>Test #[item-code]</h3>
<p>[item-field description] . . . [item-field price]</p>

[include bottom]
```

Then, to provide links to the product flypages from your home page, modify `pages/index.html` slightly, so that:

```
<td>[loop-field description]</td>
```

becomes:

```
<td><a href="[loop-code].html">[loop-field description]</a></td>
```

19.3. special_pages/missing.html

Create the `special_pages/` directory in your tutorial catalog directory (not in the `pages/` directory).

As mentioned, it is a good idea to display an error page when Interchange is asked for an unknown page. To create a missing page for display, type the following and save it as `special_pages/missing.html`.

```
[include top]
[include left]
<p>We're sorry, the page you requested has not been found.</p>

<p>Try finding what you need on the [page index]welcome page</a>.</p>
[include bottom]
```

The addition of this page ensures that users see your error message instead of a mysterious server error if they mistype a URL.

20. The shopping basket

20.1. A link for ordering

Now that you have your products available, let's add a shopping cart so customers can purchase them. This is created using the [order] [/order] tags. These tags create an HTML link that causes the specified item to be ordered and transfers the shopper to the basket page. This is a built−in shortcut to the complete order process which uses an HTML form submission process. The parameter for the [order] tag is the product ID. To add these tags to the catalog, make the following change to `pages/index.html`:

```
  <tr>
  <td>[loop−code]</td>
  <td>[loop−field description]</td>
  <td align=right>[loop−field price]</td>
+ <td>[order [loop−code]]Order Now[/order]</td>
  </tr>
  [/loop]
```

Note: The line you need to add is marked by a '+'. However, do not include the '+' when adding this line. The surrounding lines are shown to give you context. This style is called a "context diff" and is used often in this tutorial.

20.2. pages/ord/basket.html

Create the directory `pages/ord/` in the tutorial catalog directory. In other words, `ord/` should be inside the `pages/` directory.

For the [order] tag, Interchange expects a default page called `pages/ord/basket.html`. This page displays the contents of the shopping basket and contains other shopping basket functionality.

The Foundation store has a full−featured shopping basket available for use, but this tutorial teaches you to build your own simple one. The shopping basket items can be accessed using a set of tags that have an [item] prefix. Put the following code in the new file `pages/ord/basket.html`. The section that follows explains the tags used.

```
  [include top]
  [include left]

  <h2>This is your shopping cart!</h2>

  <table cellpadding=5>

  <tr>
  <th>Qty.</th>
  <th>Description</th>
  <th>Cost</th>
  <th>Subtotal</th>
  </tr>

  [item−list]
  <tr>
  <td align=right>[item−quantity]</td>
  <td>[item−field description]</td>
```

```
  <td align=right>[item-price]</td>
  <td align=right>[item-subtotal]</td>
  </tr>
  [/item-list]

  <tr><td colspan=4></td></tr>

  <tr>
  <td colspan=3 align=right><strong>Total:</strong></td>
  <td align=right>[subtotal]</td>
  </tr>

  </table>

  <hr>

  <p>
  [page checkout]Purchase now</a><br>
  [page index]Return to shopping</a>
  </p>

  [include bottom]
```

The basket items can be accessed one at a time by using the [item−list] tag. So we will create a table by iterating through the basket items. The text within the [item−list] [/item−list] tags is created for each item in the list.

- [item−quantity] shows the quantity of the item ordered. If the same item is ordered multiple times, the quantity increases.
- [item−field description] shows the description from the product database table. Any field that is not special to Interchange can be accessed from the shopping cart this way.
- [item−price] shows the per−item price that is defined in the product database table.
- [item−subtotal] shows the total cost of this order line. This is normally the price multiplied by the quantity, but it can also take into account other considerations, such as various kinds of price discounts.
- [subtotal] shows the calculated shopping basket subtotal.
- [page index] creates the starting HTML <a href="..."> for a link to the catalog welcome page.

You also need to put a link in the index page so that shoppers can go to their shopping cart without ordering something. Modify the end of `pages/index.html` by adding the following lines.

```
  </table>
+ <hr>
+ <p align=center>[page order]View shopping cart</a></p>
  [include bottom]
```

Refresh the page and test the shopping basket in your browser.

21. Order checkout

21.1. pages/checkout.html

The site can now be completed by adding the ability to check out with the shopping cart and finalize the order. To do this the customer needs to provide a shipping address (which, for the sake of this tutorial, we will assume is the same as the billing address), and payment information. We will process the order by verifying the customer's payment information and sending an email to the merchant (ourselves) detailing the order.

First you need to create a checkout page. The checkout page consists of a form that receives order information from the customer and performs a simple credit card number check. In this tutorial we will use a built−in test that only checks to see if a given credit card number could be valid. If the information is acceptable the customer will move to the next phase of the order process. If it is not, an error page will be displayed.

To create a checkout page, type the following code and save it as `pages/checkout.html`. The section that follows explains the code.

```
[include top]
[include left]
<h1>Checkout Page</h1>

<form method=post action="[process]">
<input type=hidden name=mv_todo value=submit>
<input type=hidden name=mv_order_profile value=order_profile>
<input type=hidden name=mv_cyber_mode value=minivend_test>

<table cellpadding=3>

<tr>
<td align=right><b>First name:</b></td>
<td><input type=text name=fname value="[value fname]"></td>
</tr>

<tr>
<td align=right><b>Last name:</b></td>
<td><input type=text name=lname value="[value lname]"></td>
</tr>

<tr>
<td align=right rowspan=2><b>Address:</b></td>
<td><input type=text name=address1 value="[value address1]"></td>
</tr>

<tr>
<td><input type=text name=address2 value="[value address2]"></td>
</tr>

<tr>
<td align=right><b>City:</b></td>
<td><input type=text name=city value="[value city]"></td>
</tr>

<tr>
<td align=right><b>State:</b></td>
<td><input type=text name=state value="[value state]"></td>
</tr>
```

```
<tr>
<td align=right><b>Postal code:</b></td>
<td><input type=text name=zip value="[value zip]"></td>
</tr>

<tr>
<td align=right><b>Country:</b></td>
<td><input type=text name=country value="[value country]"></td>
</tr>

</table>

<p>
Note: We assume that your billing address is the same as your shipping address.
</p>

<table cellpadding=3>

<tr>
<td align=right><b>Credit card number:</b></td>
<td><input type=text name=mv_credit_card_number value="" size=20></td>
</tr>

<tr>
<td align=right><b>Credit card expiration date:</b></td>
<td>
Month (number from 1−12):
<input type=text name=mv_credit_card_exp_month value="" size=2 maxlength=2>
<br>
Year (last two digits only):
<input type=text name=mv_credit_card_exp_year value="" size=2 maxlength=2>
</td>
</tr>

</table>

<p>
<input type=submit name=submit value="Finalize!">
<input type=reset name=reset value="Reset">
</p>

</form>

<p>[page index]Return to shopping instead</a></p>
[include bottom]
```

The HTML form begins with a method of 'post' (which sends the form data as its own stream, as opposed to the 'get' method which encodes the data as part of the URL). The [process] tag creates a special URL for form processing. Interchange has a built−in form processor that is configured by submitting certain fields in the form. The Finalize button will invoke this form processor and link the user to the `special_pages/receipt.html` page, which is described later.

You are submitting some hidden form values that will tell Interchange how to process this form. The first value, ***mv_todo*** was set as *submit*. This causes the form to be submitted for validation. The second value, ***mv_order_profile*** was set as *order_profile*. This determines the validation process for the form. It is explained further in the next section.

The last value, ***mv_cyber_mode***, was set to be *minivend_test*. The ***mv_cyber_mode*** value determines what method will be used to charge a credit card. The value of *minivend_test* uses the internal test method, which

calculates a simple checksum against the card to determine if it is a valid number.

When preparing an order for processing, Interchange looks for certain named fields in the form values for name, address, and credit card information. We are using all expected field names in this form so that no translation needs to take place.

View the checkout page in your browser. The "Finalize!" link has not been enabled, but the page should display properly.

21.2. etc/profiles.order

Create the `etc/` directory in the tutorial catalog directory now.

You need to set up verification for the order form by defining an order profile for the form. An order profile determines what fields are necessary for the form to be accepted. Create an order profile verification page by typing the following and saving it as `etc/profiles.order`. The section that follows explains the code used.

```
__NAME__ order_profile

fname=required
lname=required
address1=required
city=required
state=required
zip=required

&fatal=yes
&final=yes

__END__
```

A single file can contain multiple profile definitions. First the profile is named using the __NAME__ pragma. (This is unrelated to the __VARIABLE__ syntax seen elsewhere in Interchange.) Then in the profile there is a list of the form fields that are required. The ***&fatal*** setting indicates that validation will fail if any of the requirements are not met. ***&final*** indicates that this form will complete the ordering process. This setting is helpful if you have a multi−page ordering process and you want to validate each page individually. The __END__ pragma signals the end of this profile, after which you can begin another one.

In order to activate your order profile, add the following ***OrderProfile*** directive to the end of `catalog.cfg`:

```
OrderProfile etc/profiles.order
```

21.3. special_pages/needfield.html

If the submitted form lacks a required field, Interchange will display an error page. The default location is `special_pages/needfield.html`. To create this page, type the following text and save it as `special_pages/needfield.html`.

```
[include top]
[include left]
<p>The following information was not given:</p>
```

```
  <p><b>[error all=1 show_var=1 show_error=1 joiner='<br>']</b></p>

  <p>Please go back to the [page checkout]checkout page</a>
  and fill out the form properly.</p>

  [include bottom]
```

The [error] tag is the most important tag on this page. The ***all*** parameter tells the tag to iterate through all of the errors reported from the failed verification, and the ***show_var*** parameter indicates that the failed variable name should be displayed. For example, if the first name was left empty, *fname* would be shown. The ***show_error*** parameter displays the actual error for the variable. The ***joiner*** parameter inserts an HTML <*br*> tag between each error message, so each error is displayed on its own line. In more complex configurations, the [error] tag can be even more expressive.

21.4. Credit card processing

This tutorial uses a very simple order process. To accomplish this, one more directive needs to be added to the file `etc/profiles.order`:

```
  &fatal=yes
  &final=yes
+ &credit_card=standard keep

  __END__
```

This issues two instructions to the credit card system.

The first option, ***standard***, uses the standard built−in encryption algorithm to encrypt the credit card number and erases the unencrypted copy from memory. We are using the standard option not to encrypt the number but to run the checksum verification on the number to verify that it is a potentially correct number. We will not be checking with a real payment processor to see if it actually is a valid card number. For testing purposes, you can use the card number 4111 1111 1111 1111, which will pass the checksum test.

The second option, ***keep***, keeps the credit card number from getting removed from memory. We want to keep the number in memory so that it is available when it is mailed as part of the order.

If the credit card number passes and all of the required fields are present, the customer will be sent to the final page. Interchange then sends an e−mail to the store owner (you).

21.5. etc/report

When the customer's involvement in the order is complete, Interchange composes an email and sends it to the recipient defined in the MailOrderTo directive in `catalog.cfg`. The default location for the template for this email report is `etc/report`. Interchange tags can be used to fill in the body of the message.

The report should include at least the customer's name, address, and the items they ordered. The following is a simple report template; save it as `etc/report`.

```
              Name: [value fname] [value lname]
           Address: [value address1][if value address2]
                    [value address2][/if]
  City, State, etc.: [value city], [value state]  [value zip] [value country]
```

```
    Credit Card #: [cgi mv_credit_card_number]
  Expiration Date: [cgi mv_credit_card_exp_month]/[cgi mv_credit_card_exp_year]


********** ORDER **********
[item-list]
[item-quantity] x [item-description] ([item-code]), [item-price] ea.
[/item-list]
Subtotal: [subtotal]
   Total: [total-cost]
```

This file is in plain text format where, unlike HTML, white space is relevant. It is fairly straightforward, except that the [if] tag was added to only include the optional second address line if the customer filled it in.

One of the special properties of the *mv_credit_card_number* field is that Interchange specifically precludes the credit card number from being saved. This makes it unavailable to you in the [value] tag. The ***[cgi]*** tag is used to circumvent this important security measure in order to get the value submitted from the last form.

WARNING! Obviously it is a bad idea to send a real credit card number over an insecure channel like email. In a real configuration, you would encrypt the number securely before emailing or storing it.

21.6. special_pages/receipt.html

Once the report has been run, Interchange will finish the order process on the customer side by displaying a success screen containing a receipt. The default location for this page is `special_pages/receipt.html`. To create a receipt page, type the following code and save it as `special_pages/receipt.html`.

```
[include top]
[include left]
<p>Thank you for ordering stuff from us.<br>Have a nice day!</p>
<p>[page index]Return to our welcome page</a></p>
[include bottom]
```

Once the order is processed, the customer's shopping cart is emptied.

At this point you have a more-or-less functional store. Congratulations.

22. Enhancing the catalog

Now that you have a working catalog, you can go back and add improvements and test them incrementally. This section walks you through several and then suggests more enhancements you can attempt on your own.

22.1. Price pictures

You may have noticed that the product prices aren't formatted as prices usually are. The way to correct this is with an Interchange feature called *price pictures*.

There are several properties to price pictures: the currency symbol, the thousands separator, the decimal point, the number of digits to show behind the decimal, and so on. Most Unix systems have U.S. currency and the English language as the default locale, which is called `en_US`. The only thing you need to do on such a system is specify the currency symbol, which, in this case, is the dollar sign. To do this, add the following line to your `catalog.cfg` file:

```
Locale en_US currency_symbol $
```

Restart Interchange and view your catalog. You will notice little has changed on the welcome page or the flypages, but in the shopping cart all your prices should be formatted as U.S. dollars ("1347.3" has become "$1,347.30"). This is because Interchange automatically formats shopping cart prices as currency. To turn off this feature, you would have to change the [item−price] tag to [item−price noformat] in `pages/ord/basket.html`.

But that's probably not what you want to do. You're probably more interested in formatting your other prices as currency. To do that, simply use the [currency] [/currency] tag pair for all price values. Make the following change to `pages/index.html`:

```
  [loop search="ra=yes/fi=products"]
  <tr>
  <td>[loop−code]</td>
  <td>[loop−field description]</td>
− <td align=right>[loop−field price]</td>
+ <td align=right>[currency][loop−field price][/currency]</td>
  </tr>
  [/loop]
```

Note: The line that begins with '−' should be deleted. Do not type the '−'. The next line, that starts with '+', replaces it.

A similar change to the `[item−field price]` tag in the `pages/flypage.html` page will fix that currency display. View the page in your browser. All your prices should be formatted for U.S. currency.

If your prices are not being formatted correctly, your default system locale may be set up differently or your `en_US` locale settings may be wrong. There are a few other `catalog.cfg` directives you can use to correct the situation:

```
Locale en_US p_cs_precedes 1
```

Makes the currency symbol precede the currency value. A '0' setting makes the symbol come after the currency value.

```
Locale en_US mon_thousands_sep ,
```

Sets your thousands separator to a comma. It can be set to any value.

```
Locale en_US mon_decimal_point .
```

Sets your decimal separator to a comma. Many countries use a comma instead of a period to separate the integer from the decimal part.

Note: Consult the Interchange documentation and your operating system manual for more information on locale settings.

22.2. Catalog variables

Interchange provides a very useful feature that has not been discussed yet called catalog variables. It provides a way for you to set a variable to a certain value in the `catalog.cfg` file and use it anywhere in your catalog pages. The ***Variable*** directive allows an Interchange catalog variable to be created with the name coming from the first parameter and the value from the rest of the line, like this:

```
Variable SOMENAME whatever value you want
```

To access that variable in your pages, type the token __SOMENAME__. Notice that there are two underscore characters before the variable name and two after it, and that in place of the word SOMENAME you would put the actual name of the variable. The first thing Interchange does on a page is to replace the token with the variable's value. The value can also include Interchange tags to be parsed.

22.3. A more interesting page footer

You can put a contact email address at the bottom of each page in case your customers want to contact you. You could just add it to the footer, but by putting it into a variable you can use it in contact pages as well. This allows you to easily change the variable information and have that change reflected in all instances of that variable. The following is an example of how to set a catalog variable in `catalog.cfg`:

```
Variable CONTACT_EMAIL someone@your.domain
```

Now make the following change to your template file `bottom`:

```
  </td>
  </tr>
- <tr colspan=2><td>(bottom)</td></tr>
+ <tr colspan=2><td><a href="mailto:__CONTACT_EMAIL__">Contact us</a>
+ if you have any questions.</td></tr>
  </table>
  </div>
  </body>
  </html>
```

Be sure to restart Interchange before reloading the page in your browser, since you made a change to `catalog.cfg`.

Let's add another variable to your catalog. This variable demonstrates how an Interchange tag can be included in the variable. This Interchange tag returns the current date in a standard format. Add the following to

`catalog.cfg`:

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

Note: See the *Interchange Tag Reference Guide* for an explanation of the [time] tag.

Now add the following to the `left` template piece:

```
  <tr>
- <td align=center>(left)</td>
+ <td align=center>__DISPLAYDATE__</td>
  <td align=center>
```

Restart Interchange and view the page.

22.4. Advanced credit card expiration date selection

To reduce the possibility of human error at checkout time, most online stores use a pull−down option menu to list the months and the years for the credit card expiration date, instead of having the user to type the numbers by hand. It also lets you avoid explaining whether the user should enter a 2− or 4−digit year.

Make the following change to your `pages/checkout.html` page. The section that follows explains the code. Read the explanation section below before typing the code to be sure you know where tabs should be used instead of spaces and where to watch out for `backticks`.

```
  <tr>
  <td align=right><b>Credit card expiration date:</b></td>
  <td>
- Month (number from 1−12):
- <input type=text name=mv_credit_card_exp_month value="" size=2 maxlength=2>
- <br>
- Year (last two digits only):
- <input type=text name=mv_credit_card_exp_year value="" size=2 maxlength=2>
+
+ Month:
+ <select name=mv_credit_card_exp_month>
+ [loop
+    lr=1
+    option=mv_credit_card_exp_month
+    list="
+ 1     01 − January
+ 2     02 − February
+ 3     03 − March
+ 4     04 − April
+ 5     05 − May
+ 6     06 − June
+ 7     07 − July
+ 8     08 − August
+ 9     09 − September
+ 10    10 − October
+ 11    11 − November
+ 12    12 − December"]
+ <option value="[loop−code]">[loop−pos 1]
+ [/loop]
+ </select>
+
+ Year:
```

```
+ <select name=mv_credit_card_exp_year>
+ [comment]
+    This should always return the current year as the first, then
+    seven more years.
+ [/comment]
+ [loop option=mv_credit_card_exp_year lr=1 list=`
+   my $year = $Tag->time( '', { format => '%Y' }, '%Y' );
+   my $out = '';
+   for ($year .. $year + 7) {
+     /\d\d(\d\d)/;
+     $last_two = $1;
+     $out .= "$last_two\t$_\n";
+   }
+   return $out;
+ `]
+   <option value="[loop-code]">[loop-pos 1]
+ [/loop]
+ </select>
+
  </td>
  </tr>

  </table>
```

In the first set of <select> </select> tags a list is generated of the months to choose from. This is accomplished by using a ***[loop]*** tag. In this case we are looping over an explicit list. The list is provided in the *list* parameter. Use caution when typing this, as it is sensitive to formatting (which may not be reflected in this document). Make sure that the numbers are the first characters on each new line and that the elements are separated by a single tab. Since the columns in this list are not named, the first element can be accessed using ***[loop-code]*** or ***[loop-pos 0]*** with subsequent elements being accessed by ***[loop-pos N]*** where N is the number of the element you want. Notice that the elements are zero-indexed. Each time through this loop Interchange generates a select <option> with a number as the value and the name of the month as the text for the select menu.

For the next set of <select> </select> tags embedded Perl is used to generate the list which is iterated over. Perl code can be embedded in Interchange pages in order to extend the abilities of the system. Make sure you typed backticks (grave accents) after "list=" and before the closing bracket and not apostrophes. This code generates an entry for seven years in addition to the current year. It is not necessary at this point for you to understand this Perl code.

22.5. Sorting the product list

The products listed on your welcome page are shown in the same order that you entered them into `products/products.txt`. As you add more products, you will want this list to show up in a predictable order. To do this, you need to change the search parameters in `index.html`, which were originally:

```
[loop search="ra=yes/fi=products"]
```

You will recall that 'ra' stands for 'return all' and 'fi' stands for file. Let's add the search parameter 'tf', which specifies the sor**t f**ield. You can specify the field either by name or by number (starting with 0), with names and order as given in the first line of `products/products.txt`). Make the following change in `index.html`:

```
[loop search="ra=yes/fi=products/tf=price"]
```

Refresh your browser. The default ordering is done on a character−by−character basis, but we were looking to do a numeric sort. For this you need to set 'to', the sor**t o**rder, to 'n', for *numeric*:

```
[loop search="ra=yes/fi=products/tf=price/to=n"]
```

Refresh your browser. Now try reversing the sort order by adding 'r' to the 'to' setting:

```
[loop search="ra=yes/fi=products/tf=2/to=nr"]
```

You'll notice that it worked equally well to specify the sort field by number instead of name. You could also do a reverse alphabetical sort by description:

```
[loop search="ra=yes/fi=products/tf=1/to=r"]
```

Now let's try narrowing the search down a bit. Instead of returning all, we'll give 'se', the **se**arch paramerter, and and use 'su', which allows **su**bstring matches. To search only for products that have the word "test" in one of their fields, and sort the results by description, type:

```
[loop search="se=test/su=yes/fi=products/tf=description"]
```

Which seems like something that would be better done in a search box for your store visitors.

Before moving on, change this search back to the simple list, sorted by description:

```
[loop search="ra=yes/fi=products/tf=description"]
```

22.6. Adding a search box

Your customers might appreciate the ability to search for a test by SKU or part of the test description. To do this, you need to add a search box to the left portion of the page layout. Make the following change to the file `left`:

```
  <tr>
- <td align=center>__DISPLAYDATE__</td>
+ <td align=center>
+ <form action="[area search]" method=post>
+ Search:<br>
+ [set testname]su=yes/fi=products/sf=sku/sf=description[/set]
+ <input type=hidden name=mv_profile value=testname>
+ <input type=text name=mv_searchspec size=15 value="">
+ </form>
+ <hr>
+ __DISPLAYDATE__
+ </td>
  <td align=center>
```

This is a simple HTML form with a single input box for text. The action goes to a special Interchange processor called 'search' that will perform the search and pass the results to a page called `pages/results.html` (that has not been created yet).

The [set testname] ... [/set] tags set an Interchange 'value' variable that, in this case, will be used as a predefined search profile. We specify all the search parameters except the one the user will enter, 'mv_searchspec' (the long name for 'se'). We then tell Interchange we want to use this search profile in a

hidden form tag named 'mv_profile'.

The search box will now appear on all catalog pages, but you still need to create the search results page. To create the search results page, type the following code and save it as `pages/results.html`.

```
[include top]
[include left]
<h3>Search Results</h3>
[search-region]
  [on-match]
    <table cellpadding=5>
    <tr>
    <th>Test #</th>
    <th>Description</th>
    <th>Price</th>
    </tr>
  [/on-match]
  [search-list]
    <tr>
    <td>[item-code]</td>
    <td><a href="[item-code].html">[item-field description]</a></td>
    <td align=right>[item-field price]</td>
    <td>[order [item-code]]order now[/order]</td>
    </tr>
  [/search-list]
  [on-match]
    </table>
  [/on-match]
  [no-match]
    <p>Sorry, no matches were found for '[cgi mv_searchspec]'.</p>
  [/no-match]
[/search-region]
<hr>
<p align=center>[page index]Return to welcome page</a></p>
<p align=center>[page order]View shopping cart</a></p>
[include bottom]
```

The search results will be contained in the [search−region] [/search−region] tags. The text in the [on−match] [/on−match] container will be displayed only if matches were found for the search. The text in the [no−match] [/no−match] container will be displayed only if no matches were found. The [search−list] [/search−list] container functions just like [loop] [/loop], iterating over its contents for each item in the search results list.

22.7. The default catalog page

As you know, a standard Interchange catalog page URL looks like this:

```
http://localhost/cgi-bin/tutorial/index.html
```

But what happens if you leave off the page name, as people often do when typing URLs in by hand? Type:

```
http://localhost/cgi-bin/tutorial
```

and you get a server error message. We can change this by adding the following directive to `catalog.cfg`:

```
SpecialPage catalog index
```

Restart Interchange and try the above URL again.

Note: If you want to make the welcome page something other than `pages/index.html`, modify the 'index' part of the directive appropriately.

22.8. High−traffic changes

Through this tutorial you have created catalog pages that use the [include] tag to include template pieces in the pages. This has worked well, but there are a few drawbacks. First, if you want to rename any of the template piece files or move them out of the main catalog directory and into their own subdirectory, you would have to update the [include] tag on every page. To avoid this, you can create catalog variables set to the [include] tags. Add these lines to your `catalog.cfg` file:

```
Variable TOP    [include top]
Variable LEFT   [include left]
Variable BOTTOM [include bottom]
```

Now change every instance of [include top] to __TOP__, doing the same for each [include] tag. At this point, you might not want to do a search−and−replace on all the .html files you just created, but keep this capability in mind for the next catalog you work on.

If you made all of the replacements and then renamed and moved your `top` file, you would only have to make a single change for each region in `catalog.cfg` to get your pages up to date:

```
Variable TOP    [include templates/main-top]
```

And so on, depending on your naming scheme.

22.9. High traffic mode

Every time a catalog page is viewed, each file in an [include] tag must be loaded from disk. In a test situation, this takes no noticeable amount of time. But on a busy Interchange server, this can slow your system.

You can switch to a high−traffic mode that doesn't require each template piece to be read from disk every time the page is loaded. Instead, all of the pieces are read into variables once when Interchange is started and they remain in memory until Interchange is restarted. On very busy Interchange catalogs, this can increase your speed noticeably. The only drawback is that you need to restart the Interchange daemon when you make changes to the template pieces in order to have the changes take effect. You can set up high−traffic templates by changing the Variable directives in `catalog.cfg` as follows:

```
Variable TOP    <top
Variable LEFT   <left
Variable BOTTOM <bottom
```

23. Ideas for further enhancements

You can expand your skill with Interchange by adding more functionality to your test catalog. Here are some simple ideas to get you started:

- Send the customer a receipt by email
- Allow customer to specify item quantities
- Generate a unique order number for each order
- Store each order in a database
- Interface with GnuPG or PGP to encrypt credit card numbers in email reports
- Organize your products into categories and group lists by category

A. Catalog directory structure

This diagram shows the directory and file structure used for the 'tutorial' catalog you built. The base will be a directory with the name of your catalog:

```
tutorial/
|
|----bottom
|----catalog.cfg
|----error.log *
|----etc/
     |----profiles.order
     |----report
|----left
|----pages/
     |----checkout.html
     |----flypage.html
     |----index.html
     |----ord/
          |----basket.html
     |----results.html
|----products/
     |----products.gdbm *
     |----products.txt
|----session/
     |----(many subdirectories and files) *
|----special_pages/
     |----missing.html
     |----needfield.html
     |----receipt.html
|----tmp/ *
|----top
```

* denotes files that are automatically created by Interchange at run time. The name of `products.gdbm` may vary on your system depending on your Perl setup and default system DBM libraries.

B. Document history

October 2000. Conceived and written by Sonny Cook.

December 2000. Edited and expanded by Jon Jensen.

January 2001. Proofread and clarified by Alison Smith and David Adams.

12 January 2001. First public release.

Configuration Reference

24. Interchange Configuration Files

The Red Hat Interchange 4.8 Configuration Reference is an alphabetical reference to the configuration directives used in Interchange global and catalog configuration files.

Interchange has multiple catalog capability, and therefore splits its configuration into two pieces. One is global, `interchange.cfg`, and affects every catalog running under it. The other, `catalog.cfg` is specific to an individual catalog, and has no effect on other catalogs.

24.1. Directive syntax

Configuration directives are normally specified with the directive as the first word on the line, with its value or values following. Capitalization of the directive name is not signifigant. Leading and trailing whitespace is stripped from the line.

Including files in directives

Additional files may be called with an include file notation like this:

```
DirectiveName <includefile
```

Files included from interchange.cfg are relative to the Interchange software directory. Files included from catalog.cfg are relative to the catalog directory.

Here documents

A "here document" can be used to spread directive values over several lines, with the usual Perl <<MARKER syntax. No semicolon is used to terminate the marker. The closing marker must be the only thing on the line. No leading or trailing characters are allowed, not even whitespace. Here is a hypothetical directive using a here document:

```
DirectiveName <<EOD
    setting1
    setting2
    setting3
EOD
```

That is equivalent to:

```
DirectiveName setting1 setting2 setting3
```

Include single setting from file

Value can be pulled from a file with <file:

```
Variable MYSTUFF <file
```

This works well for includes that must be of the highest possible performance. They can be simply placed in a page with __VARIABLE__.

include

Other configuration files can be included in the current one. For example, common settings can be set in one file:

```
include common.cfg
```

Or all files in one directory:

```
include usertag/*
```

ifdef and ifndef

ifdef/endif and ifndef/endif pairs can be used:

```
Variable ORDERS_TO email_address

ifdef ORDERS_TO
ParseVariables Yes
MailOrderTo __ORDERS_TO__
ParseVariables No
endif

ifdef ORDERS_TO =~ /foo.com/
# Send all orders at foo.com to one place now
# Set ORDERS_TO to stop default setting
Variable  ORDERS_TO  1
MailOrderTo   orders@foo.com
endif

ifdef ORDERS_TO eq 'nobody@nowhere.com'
# Better change to something else, set ORDERS_TO to stop default
Variable  ORDERS_TO  1
MailOrderTo   someone@somewhere.com
endif

ifndef ORDERS_TO
#Needs to go somewhere....
MailOrderTo  webmaster@localhost
endif
```

25. interchange.cfg

The VendRoot directory, specified in the main program `interchange`, is the default location of all of the Interchange program, configuration, special, and library files. Unless changed in the call to `interchange`, the main Interchange server configuration file will be `interchange.cfg` in the VendRoot directory.

The directives defined in `interchange.cfg` affect the entire Interchange server and all catalogs running under it. Multiple Interchange servers may be run on the same machine with totally independent operation.

Following is an alphabetical listing of all global configuration directives.

25.1. ActionMap *global*

Allows setting of Interchange form actions, usually with a Perl subroutine. Actions are page names like:

```
process  Perform a processing function
order    Order items
scan     Search based on path info
search   Search based on submitted form variables
```

The global version of ActionMap applies to all catalogs. If the same action is specified in `catalog.cfg`, it will pertain. See `ActionMap` in that section.

25.2. AddDirective *global*

Adds a configuration directive that will be parsed for every `catalog.cfg` file. Accepts three parameters: the name of the directive, the name of the parser (if any), and the default value (if any). The following definition would add a directive "Foo," with parser "parse_bar," and a default value of "Hello, world!":

```
AddDirective  Foo  bar "Hello, world!"
```

If the parser is not defined, the directive value will be scalar and the same as what the user passes in the config file. If defined, the parser must be extant before it can be referenced, is always resident in Vend::Config, and begins with the string `parse_`. Examples can be found in the files in the distribution software directory `compat/`.

25.3. AdminSub *global*

Marks a global subroutine for use only by catalogs that are set to `AllowGlobal` (see below). Normally global subroutines can be referenced (in embedded Perl) by any catalog.

```
AdminSub  dangerous
```

25.4. AllowGlobal *global*

Specifies catalog identifiers that may define subroutines and UserTag entries that can operate with the full permissions of the server. **Don't use this unless the catalog user is trusted implicitly.** Default is blank.

```
AllowGlobal  simple
```

Using AllowGlobal is never necessary, and is always dangerous in a multi−user environment. Its use is not recommended.

25.5. AutoVariable *global*

Specifies directives which should be translated to Variable settings. For scalars, the directive name becomes the Variable name and yields its value, i.e. `ErrorFile` becomes `__ErrorFile__`, which would by default be `error.log`. Array variables have a `_N` added, where `_N` is the ordinal index, i.e. `SafeUntrap` becomes `__SafeUntrap_0__`, `__SafeUntrap_1__`, etc. Hash variables have a _KEY added, i.e. `SysLog` becomes `__SysLog_command__`, `__SysLog_facility__`, etc. Doesn't handle hash keys that have non−word characters or whitespace. Only single−level arrays and hashes are translated properly.

See `AutoVariable` in `catalog.cfg`.

25.6. Catalog *global*

Specifies a catalog that can run using this Interchange server. This directive is usually inserted into `interchange.cfg` by the makecat program when you build a new catalog.

There are three required parameters, as shown in this example:

```
Catalog  simple /home/interchange/simple /cgi−bin/simple
```

The first is the name of the catalog. It will be referred to by that name in error, warning, and informational messages. It must contain only alphanumeric characters, hyphens, and underscores. It is highly recommended that it be all lower case.

The second is the base directory of the catalog. If the directory does not contain a `catalog.cfg` file, the server will report an error and refuse to start.

The third is the `SCRIPT_NAME` of the link program that runs the catalog. This is how the catalog is selected for operation. Any number of alias script names may be specified as additional parameters. This allows the calling path to be different while still calling the same catalog:

```
Catalog  simple /home/interchange/simple /cgi−bin/simple /simple
```

This is useful when calling an SSL server or a members−only alias that requires a username/password via HTTP Basic authorization. All branched links will be called using the aliased URL.

The script names must be unique among CGI program paths that run on this server; the same name cannot be used for more than one catalog unless the `FullURL` directive is specified. In this case, the parameter may be specified as:

```
www.yourcompany.com/cgi−bin/simple
www.theirs.com/cgi−bin/simple
```

Each of those 'simple' catalogs would then call a different catalog.

Optionally, individual `Catalog` directives that specify each of the different parameters may be used. The equivalent of our original example directive above is:

```
Catalog simple directory /home/interchange/simple
Catalog simple script    /cgi-bin/simple
Catalog simple alias     /simple
```

Global directives may be specified that will change for that catalog only. This is mostly useful for `ErrorFile` and `DisplayErrors`:

```
Catalog simple directive ErrorFile /var/log/interchange/simple_error.log
```

25.7. CheckHTML *global*

Set to the name of an external program that will check the users HTML when they set `[flag checkhtml]` or `[tag flag checkhtml][/tag]` in their page.

```
CheckHTML  /usr/local/bin/weblint
```

25.8. ConfigAllAfter *global*

The name of a file (or files) which should be read as a part of every catalog's configuration, after any other configuration files are read. Default is `catalog_after.cfg`.

```
ConfigAllAfter   check_actions.cfg check_variables.cfg
```

25.9. ConfigAllBefore *global*

The name of a file (or files) which should be read as a part of every catalog's configuration, before any other configuration files are read. Default is `catalog_before.cfg`.

```
ConfigAllBefore   set_actions.cfg set_variables.cfg
```

25.10. ConfigParseComments *global*

Set to No if you want old−style '#include', '#ifdef', or '#ifndef' to be treated as the comments they appear to be. The default is Yes, which means both '#include' and 'include' do the same thing. (Use a space after the '#' if you really want to comment out the command.)

Interchange prior to version 4.7 used a different syntax for meta−directives 'include', 'ifdef', and 'ifndef' in configuration files. The commands were borrowed from the C preprocessor, and true to their C heritage, they started with '#': '#include', '#ifdef', '#ifndef'. Interchange configuration files, unlike C, uses '#' to begin one−line comments, which meant that a newcomer at first glance might assume that:

```
#Variable DEBUG 1
#include more.cfg
```

were both comments, when in fact the second was a live #include command.

To begin to make things more consistent, Interchange 4.7 and up now recognize those meta−directives without the leading '#', and the included demo catalog sets this directive to No so that lines beginning with '#' really are skipped as comments, regardless of what comes after.

25.11. Database *global*

Defines a database which is global and available to all catalogs. Writing can be controlled by catalog. See `Database`.

25.12. DataTrace *global*

Set DBI to trace at the level specified. Valid values are:

0 − Trace disabled.

1 − Trace DBI method calls returning with results or errors.

2 − Trace method entry with parameters and returning with results.

3 − As above, adding some high−level information from the driver and some internal information from the DBI.

4 − As above, adding more detailed information from the driver. Also includes DBI mutex information when using threaded Perl.

5 and above − As above but with more and more obscure information.

Trace level 1 is best for most Interchange debug situations. Trace will only be enabled when `DebugFile` is specified, as that file is the target for the trace. Example:

```
DataTrace   1
```

Default is 0. Directive added in 4.7.0.

25.13. DebugFile *global*

Names a file, relative to the Interchange root directory, which should store the output of `logDebug` statements, and warnings if warnings are enabled.

```
DebugFile  /tmp/icdebug
```

25.14. DisplayErrors *global*

While all errors are reported in the error log file, errors can also be displayed by the browser. This is convenient while testing a configuration. Unless this is set, the `DisplayErrors` setting in the user catalogs will have no effect. Default is No.

```
DisplayErrors       Yes
```

Note: This changes the value of $SIG{__DIE__} and may have other effects on program operation. This should NEVER be used for normal operation.

25.15. DomainTail *global*

Implements the domain/IP session qualifiers so that only the major domain is used to qualify the session ID. This is a compromise on security, but it allows non−cookie−accepting browsers to use multiple proxy servers in the same domain. Default is Yes.

```
DomainTail No
```

If encrypting credit cards with PGP or GPG, or are using a payment service like CyberCash, look at the `WideOpen` directive, which enables more browser compatibility at the cost of some security.

25.16. DumpStructure *global*

Tells Interchange to dump the structure of catalogs and the Interchange server to a file with the catalog name and the extension `.structure`. Use this to see how directives have been set.

25.17. EncryptProgram *global*

Specifies the default encryption program that should be used to encrypt credit card numbers and other sensitive information. Default is `gpg` if found on the system; then `pgpe`, if found; then `pgp`, and finally `none`, disabling encryption.

This is used to set the default in `catalog.cfg`, which has its own independent setting of `EncryptProgram`.

25.18. Environment *global*

Environment variables to inherit from the calling CGI link program. An example might be PGPPATH, used to set the directory which PGP will use to find its key ring.

```
Environment  MOD_PERL REMOTE_USER PGPPATH
```

25.19. ErrorFile *global*

Sets the name of the global error log. The default is `error.log` in the Interchange software directory.

```
ErrorFile  /var/log/interchange/log
```

Of course, the user ID running the Interchange server must have permission to write that file.

Optionally, syslog error logging can be set up as well. See `SysLog`.

25.20. FormAction *global*

Allows a form action (like the standard ones `return, submit, refresh,` etc.) to be set up. It requires a Perl subroutine as a target:

```
FormAction foo <<EOR
```

```
sub {
    $CGI->{mv_nextpage} = 'bar';
}
EOR
```

If it returns a true (non−zero, non−empty) value, Interchange will display the page defined in $CGI−>{mv_nextpage}. Otherwise, Interchange will not display any page. The default Interchange actions can be overridden, if desired. There is also a catalog−specific version of this directive, which overrides any action of the same name.

The global version affects all catalogs −− there is also a catalog−specific version of FormAction which is protected by Safe.

25.21. FullUrl *global*

Normally Interchange determines which catalog to call by determining the SCRIPT_NAME from the CGI call. This means that different (and maybe virtual) hosts cannot use the same SCRIPT_NAME to call different catalogs. Set `FullUrl` to Yes to differentiate based on the calling host. Then, set the server name in the Catalog directive accordingly, such as `yourdomain.com/cgi−bin/simple`. A yes/no directive, the default is No.

```
FullUrl  Yes
```

If it is set in this fashion, `all` catalogs must be defined in this fashion. NOTE: The individual catalog setting will not work, as this is used before the catalog name is known.

25.22. GlobalSub *global*

Defines a `global` subroutine for use by the `[perl sub] subname arg /perl]` construct. Use the "here document" capability of Interchange configuration files to make it easy to define:

```
GlobalSub <<EOF

sub count_orders {
    my $counter = new File::CounterFile "/tmp/count_orders", '1';
    my $number = $counter->inc();
    return "There have been $number orders placed.\n";
}
EOF
```

As with Perl "here documents," the EOF (or other end marker) must be the ONLY thing on the line, with no leading or trailing white space. Do not append a semicolon to the marker. (The above marker appears indented. It should not be that way in the file!)

IMPORTANT NOTE: These global subroutines are not subject to security checks. They can do most anything! For most purposes, scratch subroutines or catalog subroutines (also Sub) are better.

`GlobalSub` routines are subject to full Perl use strict checking, so errors are possible if lexical variables or complete package qualifications are not used for the variables.

25.23. HammerLock *global*

The number of seconds after which a locked session could be considered to be lost due to malfunction. This will kill the lock on the session. Only here for monitoring of session hand−off. If this error shows up in the error log, the system setup should be examined. Default is 30.

```
HammerLock          60
```

This mostly doesn't apply to Interchange when using the default file−based sessions.

25.24. HitCount *global*

Increments a counter in `ConfDir` for every access to the catalog. The file is named `hits.catalogname`, where `catalogname` is the short catalog identifier. A Yes/No directive, default is `No`.

```
HitCount  Yes
```

25.25. HouseKeeping *global*

How often, in seconds, the Interchange server will "wake up" and look for user reconfiguration requests and hung search processes. On some systems, this wakeup is the only time the server will terminate in response to a stop command. Default is 60.

```
HouseKeeping    5
```

25.26. Inet_Mode *global*

Determines whether INET−domain sockets will be monitored on startup. Overridden by the command−line parameter `−i`. Default is `Yes`.

25.27. IpHead *global*

Implements the domain/IP session qualifiers so that only the first `IpQuad` dot−quads of the IP address are used to qualify the session ID. The default is 1. This is a slight compromise on security, but it allows non−cookie−accepting browsers, like AOL's V2.0, to use multiple proxy servers.

`DomainTail` is preferable unless one of your HTTP servers does not do host name lookups. Default is `No`, and DomainTail must be set to `No` for it to operate.

```
IpHead Yes
```

25.28. IpQuad *global*

The number of dot−quads that IpHead will look at. Default is 1.

```
IpQuad  2
```

25.29. Locale *global*

Sets the global `Locale` for use in error messages. Normally set from a file's contents, as in the example before:

```
Locale <locale.error
```

25.30. LockoutCommand *global*

The name of a command (as it would be entered from the shell) that will lock out the host IP of an offending system. The IP address will be substituted for the first occurrence of the string %s. This will be executed with the user ID that Interchange runs under, so any commands that require root access will have to be wrapped with an SUID program.

On Linux, a host may be locked out with:

```
ipfwadm -I -i deny -S %s
```

This would require root permissions, however, under normal circumstances. Use `sudo` or another method to wrap and allow the command.

A script can be written which modifies an appropriate access control file, such as .htaccess for your CGI directory, to do another level of lockout. A simple command line containing `perl -0777 -npi -e 's/deny/deny from %s\ndeny/' /home/me/cgi-bin/.htaccess` would work as well (remember, the %s will become the IP address of the offending user).

```
LockoutCommand   lockout %s
```

25.31. LockType *global*

Allows selection of file locking method used throughout Interchange. Options are 'flock', 'fcntl', and 'none'. Added in 4.7.0.

Default is flock. See the flock(2) manpage for details.

The fcntl setting is needed for NFS filesystems; for NFS-based locking to work, the NFS lock daemon (lockd) must be enabled and running on both the NFS client and server. Locking with fcntl works on Linux and should work on Solaris, but is not guaranteed to work on all OSes.

The none setting turns off file locking entirely, but that is never recommended. It might be useful to check if locking is causing hangs on the system.

If you are only accessing sessions on an NFS-mounted directory but the rest of Interchange is on the local filesystem, you can instead set the SessionType catalog directive to 'NFS', which enables fcntl locking for sessions only on a per-catalog basis.

25.32. Mall *global*

Set to `Yes` to issue cookies only for the current catalog's script. By default, when Interchange issues a cookie it does so for the base domain. This will allow multiple catalogs to operate on the same domain without interfering with each others session ID.

A yes/no directive.

```
Mall   Yes
```

25.33. MaxServers *global*

The maximum number of servers that will be spawned to handle page requests. If more than MaxServers requests are pending, they will be queued (within the defined capability of the operating system, usually five pending requests) until the number of active servers goes below that value.

```
MaxServers     4
```

Default is 10.

25.34. NoAbsolute *global*

Whether Interchange `[file ...]` and other tags can read any file on the system (that is readable by the user id running the Interchange daemon). The default is No, which allows any file to be read. This should be changed in a multi−user environment to minimize security problems.

```
NoAbsolute     Yes
```

25.35. PIDcheck *global*

If non−zero, enables a check of running Interchange processes during the housekeeping routine. If a process has been running (or is hung) for longer than PIDcheck seconds then a kill −9 will be issued and the server count decremented. During the housekeeping routine, the number of servers checked by `MaxServers` will be recounted based on PID files.

Default is 0, disabling the check.

```
PIDcheck   300
```

If have long−running database builds, this needs to be disabled. Set it to a high value (perhaps 600, for 10 minutes), or use the offline script.

25.36. PIDfile *global*

The file which will contain the Interchange server process ID so that it can be read to determine which proces should be sent a signal for stopping or reconfiguring the server.

```
PIDfile  /var/run/interchange/interchange.pid
```

This file must be writable by the Interchange server user ID.

25.37. Profiles *global*

Names a file (or files) which contain `OrderProfile` and `SearchProfile` settings that will apply for all catalogs.

```
Profiles     etc/profiles.common
```

25.38. SafeUntrap *global*

Sets the codes that will be untrapped in the `Safe.pm` module and used for embedded Perl and conditional operations. View the Safe.pm documentation by typing `perldoc Safe` at the command prompt. The default is `ftfile sort`, which untraps the file existence test operator and the sort operator. Define it as blank to prevent any operators but the default restrictive ones.

```
SafeUntrap     ftfile sort ftewrite rand
```

25.39. SendMailProgram *global*

Specifies the program used to send email. Defaults to '/usr/lib/sendmail'. If it is not found at startup, Interchange will return an error message and refuse to start.

```
SendMailProgram     /bin/mailer
```

A value of 'none' will disable the sending of emailed orders. Orders must be read from a tracking file, log, or by other means.

25.40. SOAP *global*

If set to Yes, allows handling of SOAP rpc requests.

25.41. SOAP_Host

The list of hosts that are allowed to connect to for SOAP rpc requests. Default is `localhost 127.0.0.1`.

25.42. SOAP_MaxRequests

The maximum number of requests a SOAP rpc server will handle before it commits suicide and asks for a replacement server. This prevents runaway memory leaks.

25.43. SOAP_Perms

The permissions that should be set on a SOAP UNIX−domain socket. Default is `0660`, which allows only programs running as the same UID as Interchange to access the socket.

25.44. SOAP_Socket

A list of sockets which should be monitored for SOAP requests. If they fit the form NNN.NNN.NNN.NNN:PPPP, they are IP addresses and ports for monitoring INET−domain sockets, any other pattern is assumed to be a file name for monitoring in the UNIX domain.

```
SOAP_Socket 12.23.13.31:7770 1.2.3.4:7770 /var/run/interchange/soap
```

25.45. SOAP_StartServers

The number of SOAP servers which should be started to handle SOAP requests. Default is 1.

```
SOAP_StartServers   10
```

25.46. SocketFile *global*

The name of the file which is used for UNIX−domain socket communications. Must be in a directory where the Interchange user has write permission.

```
SocketFile  /var/run/interchange/interchange.socket
```

Default is `etc/socket` or the value of the environment variable MINIVEND_SOCKET. If set, it will override the environment. It can be set on the command line as well:

```
bin/interchange −r SocketFile=/tmp/interchange.socket
```

25.47. SocketPerms *global*

The permissions (prepend a 0 to use octal notation) that should be used for the UNIX−domain socket. Temporarily set this to 666 on the command line to debug a permission problem on `vlink`.

```
bin/interchange −r SocketPerms=0666
```

25.48. StartServers

The number of Interchange page servers which should be started to handle page requests when in `PreFork` mode. Default is 1.

```
SOAP_StartServers   10
```

25.49. SubCatalog *global*

Allows definition of a catalog which shares most of the characteristics of another catalog. Only the directives that are changed from the base catalog are added. The parameters are: 1) the catalog ID, 2) the base catalog ID, 3) the directory to use (typically the same as the base catalog), and 4) the SCRIPT_NAME that will trigger the catalog. Any additional parameters are aliases for the SCRIPT_NAME.

The main reason that this would be used would be to conserve memory in a series of stores that share most of the same pages or databases.

```
SubCatalog   sample2 sample /usr/catalogs/sample /cgi-bin/sample2
```

25.50. SysLog *global*

Set up syslog(8) error logging for Interchange.

```
SysLog  command  /usr/bin/logger
SysLog  tag      int1
SysLog  alert    local3.warn
SysLog  warn     local3.info
SysLog  info     local3.info
SysLog  debug    local3.debug
```

This would cause global errors to be logged with the command:

```
/usr/bin/logger -t int1 -p local3.alert
```

and cause system log entries something like:

```
Oct 26 17:30:11 bill int1: Config 'co' at server startup
Oct 26 17:30:11 bill int1: Config 'homefn' at server startup
Oct 26 17:30:11 bill int1: Config 'simple' at server startup
Oct 26 17:30:11 bill int1: Config 'test' at server startup
Oct 26 17:30:13 bill int1: START server (2345) (INET and UNIX)
```

This would work in conjunction with a UNIX syslogd.conf entry of:

```
        # Log local3 stuff to Interchange log
        local3.*                /var/log/interchange.log
```

A custom wrapper can be created around it to get it to behave as desired. For instance, if you didn't want to use syslog but instead wanted to log to a database (via DBI), you could create a Perl script named "logdatabase" to log things:

```
    #!/usr/bin/perl

    my $script_name = "logdatabase";
    use DBI;
    use Getopt::Std;

    getopts('d:p:T:k:')
        or die "$script_name options: $@\n";

    use vars qw/$opt_d $opt_p $opt_T $opt_k/;

    my $dsn   = $opt_d || $ENV{DBI_DSN};
    my $template = $opt_T
        || "insert into log values ('~~KEY~~', '~~LEVEL~~', '~~MSG~~')";

    my $dbh = DBI->connect($dsn)
        or die "$script_name cannot connect to DBI: $DBI::errstr\n";

    my %data;

    $data{KEY} = $opt_k || '';

    local ($/);
```

```
$data{MSG} = <>;

$data{LEVEL} = $opt_p || 'interchange.info';

$template =~ s/\~\~(\w+)\~\~/$dbh->quote($data{$1})/;

my $sth = $dbh->prepare($template)
    or die "$script_name error executing query: $template\n";

$sth->execute()
    or die "$script_name error executing query: $template\n";

exit;
```

25.51. TcpHost *global*

When running in INET mode, using `tlink`, specifies the hosts that are allowed to send/receive transactions from any catalog on this Interchange server. Can be either an name or IP number, and multiple hosts can be specified in a space−separated list. Default is localhost.

```
TcpHost         localhost secure.domain.com
```

25.52. TcpMap *global*

When running in INET mode, using `tlink` or the internal HTTP server, specifies the port(s) which will be monitored by the Interchange server. Default is 7786.

To use the internal HTTP server (perhaps only for password−protected queries), a catalog may be mapped to a port. If three catalogs were running on the server `www.akopia.com`, named `simple`, `sample`, and `search`, the directive might look like this:

```
TcpMap    7786 −  7787 simple 7788 sample 7789 search
```

Note: To map large numbers of ports, use the <<MARKER here document notation in interchange.cfg. With this in effect, the internal HTTP server would map the following addresses:

```
*:7786   mv_admin
*:7787   simple
*:7788   sample
*:7789   search
```

Note: This does not pertain to the use of `tlink`, which still relies on the CGI SCRIPT_PATH. To enable this, the SCRIPT_PATH aliases /simple, /sample, etc. must be set in the `Catalog` directive. This would look like:

```
Catalog  simple  /home/interchange/catalogs/simple /cgi−bin/simple /simple
```

To bind to specific IP addresses, add them in the same fashion that they would as an Apache Listen directive:

```
TcpMap <<EOF
    127.0.0.1:7786       −
    www.akopia.com:7787  −
EOF
```

Note: As usual, the EOF should be at the beginning of a line with no leading or trailing whitespace.

25.53. TemplateDir *global*

Sets a directory which will be searched for pages if not found in the user's `pages` directory. Interchange uses this; use it to supply some default pages so the user will not have them in their directory.

```
TemplateDir    /usr/local/interchange/default_pages
```

The user's page, if it exists, will take precedence. There is also a catalog−specific version of this directive. If a page is found in that directory (or directories), it will take precedence.

25.54. TolerateGet *global*

Set to 'Yes' to enable parsing of both GET data and POST data when a POST has been submitted. The default is 'No', which means that GET data is ignored during a POST. Unfortunately this has to be a global setting because at URL parse time, the Interchange daemon doesn't yet know which catalog it's dealing with (due to catalog aliases, etc.).

25.55. UrlSepChar *global*

Sets the character which separates form parameters in Interchange−generated URLs. Default is `&`.

25.56. Unix_Mode *global*

Determines whether the UNIX−domain socket will be monitored on startup. Overridden by the command−line parameter `−u`. Default is `Yes`.

25.57. UserTag *global*

This defines a UserTag which is global in nature, meaning not limited by the `Safe.pm` module, and is is available to all Interchange catalogs running on the server. Otherwise, this is the same as a catalog UserTag.

25.58. Variable *global*

Defines a global variable that will be available in all catalogs with the notation @@VARIABLENAME@@. Variable identifiers must begin with a capital letter, and can contain only word characters (**A−Z,a−z,0−9** and underscore). They are case−sensitive. If using the `ParseVariables` directive, only variables in ALL CAPS will be parsed. These are substituted first in any Interchange page, and can contain any valid Interchange tags including catalog variables.

```
Variable   DOCUMENT_ROOT   /usr/local/etc/httpd/htdocs
```

If a variable is called with @_VARIABLE_@, and there is no catalog Variable with its name, the global Variable value will be inserted.

There are several standard variables which should not be used:

MV_FILE

Name of the last file read in, as in `[file ...]` or an externally located perl routine.

MV_NO_CRYPT

Set this to 1 to disable encrypted passwords for the `AdminUser`.

MV_PAGE

Name of the last page read in, as in the page called with mv_nextpage or mv_orderpage.

CURRENCY, MV_CURRENCY

The current locale for currency.

LANG, MV_LANG

The current locale for language.

25.59. VarName *global*

Sets the names of variables that will be remapped to and from the URL when Interchange writes it. For instance, to display the variable `mv_session_id` as `session` in the users URL:

```
VarName   mv_session_id  session
```

The default can also be set in the `etc/varnames` file after the first time Interchange is run. Setting it in `interchange.cfg` is probably better for clarity.

There is also a catalog−specific version of this setting.

26. catalog.cfg

Each catalog must have a `catalog.cfg` file located in its base catalog directory. It contains most of the configurable parameters for Interchange. Each is independent from catalog to catalog.

Additional configuration techniques are available in the `catalog.cfg` file. First, set a `Variable` and use its results in a subsequent configuration setting if `ParseVariables` is on:

```
Variable   SERVER_NAME  www.akopia.com
Variable   CGI_URL      /cgi-bin/demo

ParseVariables Yes
VendURL    http://__SERVER_NAME____CGI_URL__
ParseVariables No
```

Define subroutine watches

Almost any configuration variable can be set up to be tied to a subroutine if the Tie::Watch module is installed. It uses a notation like the <<HERE document, but <&HERE is the notation. See Interchange Programming for details.

26.1. Programming Watch Points in catalog.cfg

Almost any configuration variable can be set up to be tied to a subroutine if the `Tie::Watch` module installed. It uses a notation like the <<HERE document, but <&HERE is the notation. Here is a simple case:

```
MailOrderTo orders@akopia.com
MailOrderTo <&EOF
sub {
    my($self, $default) = @_;
    if($Values->{special_handling}) {
        return 'vip@akopia.com';
    }
    else {
        return $default;
    }
}
EOF
```

When the order is mailed out, if the user has a variable called `special_handling` set in their session (from UserDB, perhaps), the order will be sent to 'vip@akopia.com.' Note the single quotes to prevent problems with the @ sign. Otherwise, the order will get sent to the previously defined value of `orders@akopia.com`.

If the configuration value being watched is a SCALAR, the subroutine gets the following call:

```
&{$subref}(SELF, PREVIOUS_VALUE)
```

The subroutine should simply return the proper value.

SELF is a reference to the Tie::Watch object (read its documentation for what all it can do) and `PREVIOUS_VALUE` is the previously set value for the directive. If set after the watch is set up, it will simply have the effect of destroying the watch and having unpredictable effects. (In the future, a "Store" routine may

be able to be set up that can subsequently set values).

If the configuration value being watched is an ARRAY, the subroutine gets the following call:

```
&{$subref}(SELF, INDEX, PREVIOUS_VALUE)
```

INDEX is the index of the array element being accessed. Setting up watch points on array values is not recommended. Most Interchange subroutines call arrays in their list context, and no access method is provided for that.

If the configuration value being watched is a HASH, the subroutine gets the following call:

```
&{$subref}(SELF, KEY, PREVIOUS_VALUE)
```

KEY is the index into the hash, an example of HASH type Interchange configuration values. NOTE: The following is not recommended for performance reasons. The Variable is a commonly used thing and should not bear the extra overhead of tieing, but it illustrates the power of this operation:

```
Variable TESTIT Unwatch worked.

Variable <&EOV
sub {
    my ($self, $key, $orig) = @_;
    if($key eq 'TESTIT') {
        # only the first time
        if($Scratch->{$key}++) {
            $self->Unwatch();
            return $orig->{TESTIT};
        }
        else {
            return "Tie::Watch works! -- name=$Values->{name}";
        }
    }
    else {
        return $orig->{$key};
    }
}
EOV
```

The first time __TESTIT__ is called for a particular user, it will return the string "Tie::Watch works! -- name=" along with their name set in the session (if that exists). Any other variables will receive the value that they were set to previously. Once the TESTIT key has been accessed for that user, the watch is dropped upon the next access.

26.2. Configuration Directives in catalog.cfg

All directives except MailOrderTo and VendURL have default values and are optional, though most catalogs will want to configure some of them.

26.3. ActionMap

Allows setting of Interchange actions, usually with a Perl subroutine. Actions are page names like:

```
process  Perform a processing function
```

```
order    Order items
scan     Search based on path info
search   Search based on submitted form variables
```

These are the standard supplied actions for Interchange. They can be overwritten with user−defined versions if desired. For example, to ignore the `order` action, set:

```
ActionMap  order  sub { return 1 }
```

When the leading part of the incoming path is equal to `order`, it will trigger an action. The page name will be shifted up, and the `order` stripped from the page name. So this custom `order` action would essentially perform a no−op, and a URL like:

```
<A HREF="[area order/nextpage]"> Go to the next page </A>
```

would be the equivalent of "[area nextpage]." If the action does not return a true (non−zero, non−blank) status, no page will be displayed by Interchange, not even the special `missing` page. A response may also be generated via Perl or MVASP.

The standard `process` action has a number of associated `FormAction` settings. Besides using Perl, Interchange tags may be used in an action, though they are not nearly as efficient.

26.4. AlwaysSecure

Determines whether checkout page operations should always be secure. Set it to the pages that should always be secure, separated by spaces and/or tabs.

```
AlwaysSecure    ord/checkout
```

26.5. AsciiTrack

A file name to log formatted orders in. Unless preceded by a leading '/', will be placed relative to the catalog directory. Disabled by default.

```
AsciiTrack     etc/tracking.asc
```

If a `Route` is set up to `supplant`, this is ignored.

26.6. Autoend

Sets an action that is automatically performed at the end every access. It is performed after any page parsing occurs, just before the transaction ends. See `Autoload`.

26.7. Autoload

Sets an action that is automatically performed for every access. It is performed before any page parsing occurs, and before the action or page is even determined. Can contain ITL tags or a global subroutine name. If the return value is true, a normal display of $CGI−>{mv_nextpage} will occur −− if it returns a false (zero, undef, or blank) value, no page will be processed.

As an example, to remap any mv_nextpage accesses to the private subdirectory of pages, set:

```
Autoload   [perl] $CGI->{mv_nextpage} =~ s:^private/:public/:; [/perl]
```

26.8. AutoModifier

Sets an attribute in a shopping cart entry to the field of the same name in the ProductsFile pertaining to this item. This is useful when doing shipping calculations or other embedded Perl that is based on item attributes. To set whether an item is defined as "heavy" and requires truck shipment, set:

```
AutoModifier  heavy
```

When an item is added to the shopping cart using Interchange's routines, the heavy attribute will be set to the value of the heavy field in the products database. In the default demo that would be products. Any changes to ProductFiles would affect that, of course.

Some values are used by Interchange and are not legal:

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

26.9. AutoVariable

Specifies directives which should be translated to Variable settings. For scalars, the directive name becomes the Variable name and yields its value, i.e. DescriptionField becomes __DescriptionField__, which would by default be description. Array variables have a _N added, where _N is the ordinal index, i.e. ProductFiles becomes __ProductFiles_0__, __ProductFiles_1__, etc. Hash variables have a _KEY added, i.e. SpecialPage becomes __SpecialPage_missing__, __SpecialPage_violation__, etc. Doesn't handle hash keys that have non−word characters or whitespace. Only single−level arrays and hashes are translated properly.

26.10. CommonAdjust

Settings for Interchange pricing. See Chained pricing.

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

26.11. ConfigDir

The default directory where directive values will be read from when using the <file notation. Default is config. The name is relative to the catalog directory unless preceded by a /.

```
ConfigDir      variables
```

This can be changed several times in the catalog.cfg file to pick up values from more than one directory. Another possibility is to use a Variable setting to use different templates based on a setting:

```
Variable   TEMPLATE   blue

ParseVariables Yes
ConfigDir  templates/__TEMPLATE__
ParseVariables No
Variable   MENUBAR   <menubar
Variable   LEFTSIDE  <leftside
Variable   BOTTOM    <bottom
ConfigDir config
```

This will pick the `templates/blue` template. If TEMPLATE is set to `red`, it would read the variables from `templates/red`.

26.12. CookieDomain

Allows a domain to be set so that multiple servers can handle traffic. For example, to use server addresses of `secure.yourdomain.com` and `www.yourdomain.com`, set it to:

```
CookieDomain    .yourdomain.com
```

More than one domain can be set. It must have at least two periods or browsers will ignore it.

26.13. CookieLogin

Allows users to save their username/password (for Vend::UserDB) in a cookie. Expiration is set by SaveExpire and is renewed each time they log in. To cause the cookie to be generated originally, the CGI variable `mv_cookie_password` or `mv_cookie_username` must be set. The former causes both username and password to be saved; the latter just the username.

```
CookieLogin  Yes
```

Default is `No`.

26.14. Cookies

Determines whether Interchange will send (and read back) a cookie to get the session ID for links that go outside the catalog. Allows arbitrary HREF links to be placed in Interchange pages, while still saving the contents of the session. The default is Yes.

```
Cookies         Yes
```

If the Cookies directive is enabled, and `mv_save_session` is set upon submission of a user form (or in the CGI variables through a Perl `GlobalSub`), the cookie will be persistent for the period defined by `SaveExpire`.

Note: This should almost always be "Yes."

Caching, timed builds, and static page building will never be in effect unless this directive is enabled.

26.15. CreditCardAuto

If set to Yes, enables the automatic encryption and saving of credit card information. In order for this to work properly, the `EncryptProgram` directive must be set to properly encode the field. The best way to set `EncryptProgram` is with PGP in the ASCII armor mode. This option uses the following standard fields on Interchange order processing forms:

`mv_credit_card_number`

The actual credit card number, which will be wiped from memory after checking to see if it is a valid Amex, Visa, MC, or Discover card number. This variable will never be carried forward in the user session.

`mv_credit_card_exp_all`

The expiration date, as a text field in the form MM/YY (will take a four−digit year as well). If it is not present, the fields `mv_credit_card_exp_month` and `mv_credit_card_exp_year` are looked at. It is set by Interchange when the card validation returns, if not previously set.

`mv_credit_card_exp_month`

The expiration date month, used if the `mv_credit_card_exp_all` field is not present. It is set by Interchange when the card validation returns, if not previously set.

`mv_credit_card_exp_year`

The expiration date year, used if the `mv_credit_card_exp_all` field is not present. It is set by Interchange when the card validation returns, if not previously set.

`mv_credit_card_error`

Set by Interchange to indicate the error if the card does not validate properly. The error message is not too enlightening if validation is the problem.

`mv_credit_card_force`

Set this value to 1 to force Interchange to encrypt the card despite its idea of validity. Will still set the flag for validity to 0 if the number/date does not validate. Still won't accept badly formatted expiration dates.

`mv_credit_card_separate`

Set this value to 1 to cause Interchange encrypt only the card number and not accompany it with the expiration date and card type.

`mv_credit_card_info`

Set by Interchange to the encrypted card information if the card validates properly. If PGP is used in ASCII armor mode, this field can be placed on the order report and embedded in the order email, replete with markers. This allows a secure order to be read for content, without exposing the credit card number to risk.

`mv_credit_card_valid`

Set by Interchange to true, or 1, if the the card validates properly. Set to 0 otherwise.

PGP is recommended as the encryption program, though remember that U.S. commercial organizations may require a license for RSA. Interchange will work with GPG, the Gnu Privacy Guard.

```
CreditCardAuto     Yes
```

26.16. CustomShipping

If not blank, causes an error log entry if the shipping file entry is not found. Not otherwise used for shipping. See `SHIPPING` for how to go about doing that.

```
CustomShipping      Yes
```

26.17. Database

Definition of an arbitrary database, in the form "Database database file type," where "file" is the name of an ASCII file in the same format as the products database. The file is relative to VendRoot. Records can be accessed with the `[data database field key]` tag. Database names are restricted to the alphanumeric characters (including the underscore), and it is recommended that they be either all lower or all upper case. See `DATABASES`.

```
Database      reviews  reviews.txt  CSV
```

26.18. DatabaseDefault

Defines default parameters for a database. This can be used to set a default WRITE_CONTROL setting, set a default USER or PASSWORD, etc. It accepts any scalar setting, which means all **except**:

ALTERNATE_* BINARY COLUMN_DEF DEFAULT FIELD_ALIAS FILTER_* NAME NUMERIC POSTCREATE WRITE_CATALOG

This default setting is made when the table is initially defined, i.e. explicit settings for the database itself override the defaults set.

```
DatabaseDefault      WRITE_CONTROL   1
DatabaseDefault      WRITE_TAGGED    1
```

This setting must be made *before* the database is defined. To reset its value to empty, use the `Replace` directive.

```
Replace DatabaseDefault
```

26.19. DefaultShipping

This sets the default shipping mode by initializing the variable `mv_ship_mode`. If not set in `catalog.cfg`, it is default.

```
DefaultShipping     UPS
```

Somewhat deprecated, the same thing can be achieved with:

```
ValuesDefault   mv_shipmode UPS
```

26.20. DescriptionField

The field that will be accessed with the `[item-description]` element.

```
DescriptionField    description
```

Default is `description`. It is not a fatal error if this field does not exist. This is especially important for on-the-fly items. If there is an attribute set to the same name as `DescriptionField`, this will be used for display.

26.21. DirConfig

`DirConfig` allows you to batch-set a bunch of variables from files. The syntax:

DirConfig directive-name directory-glob

`directive-name` is usually `Variable`, but could be any hash-based directive. (No other standard directives currently make sense to set this way.)

`directory-glob` is a filespec that could encompass multiple directories. Files are ignored.

The directories are read for file *names* that contain only word characters, i.e. something that would be a valid Variable. (This alone might make it not suitable for other uses, but picking up the junk from the in-directory-backup-file people would be intolerable.)

Then the contents of the file is used to set the variable of the file name.

The source file name is kept in $Vend::Cfg->{DirConfig}{Variable}{VARNAME}, for use if dynamic_variables Pragma is set.

Pragma `dynamic_variables` enables dynamic updating of variables from files. Pragma `dynamic_variables_files_only` restricts dynamic variables to files only -- otherwise variables are dynamically read from the VarDatabase definition as well.

With dynamic variables, all @_VARIABLE_@ and __VARIABLE__ settings are checked first to see if the source file is defined. If there is a key present, even if its contents are blank, it is returned. Example: in the case of this catalog.cfg entry:

DirConfig Variable templates/foundation/regions

If the file NOLEFT_TOP is present at catalog config time, __NOLEFT_TOP__ will equal `[include templates/foundation/regions/NOLEFT_TOP]`.

26.22. DisplayErrors

If the administrator has enabled DisplayErrors globally, setting this to "Yes" will display the error returned from Interchange in case something is wrong with embedded Perl programs, tags, or Interchange itself. Usually, this will be used during development or debugging. Default is No.

```
DisplayErrors       Yes
```

26.23. DynamicData

When set to one or more Interchange database identifiers, any pages using data items from the specified database(s) will not be cached or built statically. This allows dynamic updating of certain arbitrary databases (even the products database) while still allowing static/cached page performance gains on pages not using those data items.

```
DynamicData         inventory
```

Overridden by `[tag flag build][/tag]`, depending on context.

26.24. EncryptProgram

Contains a program command line specification that indicates how an external encryption program will work. Two placeholders, `%p` and `%f`, are defined, which are replaced at encryption time with the password and temporary file name respectively. See `Order Security`. This is separate from the PGP directive, which enables PGP encryption of the entire order.

If PGP is the encryption program (Interchange determines this by searching for the string `pgp` in the command string), no password field or file field need be used. The field `mv_credit_card_number` will never be written to disk in this case.

```
EncryptProgram      /usr/local/bin/pgp −feat sales@company.com
```

If the order `Route` method of sending orders is used (default in the demo), this sets the default value of the `encrypt_program` attribute.

26.25. ErrorFile

This is where Interchange will write its runtime errors for THIS CATALOG ONLY. It can be shared with other catalogs or the main Interchange error log, but if it is root−based, permission to write the file is required.

```
ErrorFile   /home/interchange/error.log
```

26.26. ExtraSecure

Disallows access to pages which are marked with `AlwaysSecure` unless the browser is in HTTPS mode. A Yes/No directive, the default is 'No.'

```
ExtraSecure  Yes
```

26.27. Filter

Assigns one or more filters (comma separated) to be automatically applied to a variable.

As an example, multiple form variable submissions on the same page come back null−separated, like 'value1\0value2\0value3'. To automatically change those nulls to spaces, you could use this directive:

```
Filter  mail_list  null_to_space
```

Of course you could just as easily use the regular [filter] tag on the page if the filter is only going to be used in a few places.

See the ictags document for more information, including a complete list of filters.

26.28. FormAction

Allows set up of a form action (like the standard ones `return, submit, refresh,` etc.). It requires a Perl subroutine as a target:

```
FormAction foo <<EOR
sub {
    $CGI−>{mv_nextpage} = 'bar';
}
EOR
```

If it returns a true (non−zero, non−empty) value, Interchange will display the page defined in $CGI−>{mv_nextpage}. Otherwise, Interchange will not display any page. The default Interchange actions can be overridden if desired. There is also a global version of this directive, which is overridden if a catalog−specific action exists.

26.29. FormIgnore

Set to the name(s) of variables that should not be carried in the user session values. Must match exactly and are case sensitive.

```
FormIgnore    mv_searchtype
```

26.30. FractionalItems

Whether items in the shopping cart should be allowed to be fractional, i.e., 2.5 or 1.25. Default is No.

```
FractionalItems     Yes
```

26.31. Glimpse

The pathname for the `glimpse` command, used if `glimpse` searches are to be enabled. To use `glimpseserver`, the `−C`, `−J`, and `−K` tags must be used.

```
Glimpse  /usr/local/bin/glimpse −C −J srch_engine −K2345
```

26.32. History

How many of the most recent user clicks should be stored in the session history. Default is 0.

26.33. HTMLsuffix

The file extension that will be seen as a page in the `pages` directory. Default is `.html`.

```
HTMLsuffix .htm
```

26.34. ImageAlias

Aliases for images, ala Apache/NCSA, ScriptAlias, and Alias directives. Relocates images based in a particular directory to another for Interchange use; operates after `ImageDir`. Useful for editing Interchange pages with an HTML editor. Default is blank.

```
ImageAlias  /images/  /thiscatalog/images/
```

26.35. ImageDir

The directory where all relative IMG and INPUT source file specifications are based. IT MUST HAVE A TRAILING / TO WORK. If the images are to be in the `DocumentRoot` (of the HTTP server or virtual server) subdirectory images, for example, use the `ImageDir` specification '/images/'. This would change SRC="order.gif" to SRC="/images/order.gif" in IMG and INPUT tags. It has no effect on other SRC tags.

```
ImageDir /images/
```

Can be set in the Locale settings to allow different image sets for different locales (MV3.07 and up).

26.36. ImageDirInternal

A value for `ImageDir` only when the internal HTTP server is in use. It must have a trailing / to work, and should always begin with a fully−qualified path starting with `http://`.

```
ImageDirInternal http://www.server.name/images/
```

26.37. ImageDirSecure

A value for `ImageDir` only when the pages are being served via HTTPS. It must have a trailing / to work, and should always begin with a fully−qualified path starting with `http://`.

```
ImageDirSecure   /secure/images/
```

This is useful if using separate HTTPS and HTTP servesr, and cannot make the image directory path heads match.

26.38. Locale

Sets the special locale array. Tries to use POSIX `setlocale` based on the value of itself, then tries to accept a custom setting with the proper definitions of `mon_decimal_point`, `thousands_sep`, and `frac_digits`, which are the the only international settings required. Default, if not set, is to use US−English settings.

Example of the custom setting:

```
Locale     custom mon_decimal_point , mon_thousands_sep . frac_digits 0
```

Example of POSIX setlocale for France, if properly aliased:

```
Locale     fr
```

See `setlocale(3)` for more information. If embedded Perl code is used to sort search returns, the `setlocale()` will carry through to string collation.

See Internationalization.

26.39. LocaleDatabase

Set to the Interchange database identifier of a table that contains `Locale` settings. These settings add on to and overwrite any that are set in the catalog configuration files, including any include files.

```
Database       locale  locale.asc  TAB
LocaleDatabase locale
```

26.40. MailOrderTo

Specifies the e−mail address to mail completed orders to.

```
      MailOrderTo  orders@xyzcorp.com
```

If 'none' is specified, no e−mailed order will be sent.

26.41. NoCache

The names of Interchange pages that are not to be built statically if STATIC PAGE BUILDING is in use. If the name is a directory, no pages in that directory (or any below it) will be cached or built statically.

```
NoCache    ord
NoCache    special
```

26.42. NoImport

When set to one or more Interchange database identifiers, those database(s) will never be subject to import. Useful for SQL databases or databases that will "never" change.

```
NoImport   inventory
```

26.43. NonTaxableField

The name of the field in the products database that is set (to 1 or Yes) if an item is not to be taxed. Interchange will log an error and tax it anyway if the field doesn't exist in the database. Blank by default, disabling the feature.

```
NonTaxableField    wholesale
```

26.44. OfflineDir

The location of the offline database files for use with the Interchange offline database build command. Set to "offline" as the default, and is relative to VendRoot if there is no leading slash.

```
OfflineDir          /usr/data/interchange/offline
```

26.45. OnFly

Enables on−the−fly item additions to the shopping cart. If set to the name of a valid UserTag, that tag definition will be used to parse and format the item with the following call:

```
$item = Vend::Parse::do_tag($Vend::Cfg->{OnFly},
                            $code,
                            $quantity,
                            $fly[$j],
                          );
```

$fly[$j] is the value of `mv_order_fly` for that item. An `onfly` tag is provided by Interchange. See <On−the−fly> ordering.

26.46. OrderCounter

The name of the file (relative to catalog root if no leading /) that maintains the order number counter. If not set, the order will be assigned a string based on the time of the order and the user's session number.

```
OrderCounter       etc/order.number
```

Bear in mind that Interchange provides the order number as a convenience for display, and that no internal functions depend on it. Custom order number routines may be defined and used without fear of consequences.

If a `Route` is set up to `supplant` and the `counter` attribute is set there, this is ignored.

26.47. OrderLineLimit

The number of items that the user is allowed to place in the shopping cart. Some poorly−mannered robots may "attack" a site by following all links one after another. Some even ignore any `robots.txt` file that may have been created. If one of these bad robots orders several dozen or more items, the time required to save and restore the shopping cart from the user session may become excessive.

If the limit is exceeded, the command defined in the Global directive `LockoutCommand` will be executed and the shopping cart will be emptied. The default is 0, disabling the check. Set it to a number greater than the number of line items a user is ever expected to order.

```
OrderLineLimit   50
```

26.48. OrderProfile

Allows an unlimited number of profiles to be set up, specifying complex checks to be performed at each of the steps in the checkout process. The files specified can be located anywhere. If relative paths are used, they are relative to the catalog root directory.

```
OrderProfile    etc/profiles.order etc/profiles.login
```

The actions defined here are also used for `mv_click` actions if there is no action defined in `scratch` space. They are accessed by setting the `mv_order_profile` variable to the name of the order profile. Multiple profiles can reside in the same file, if separated by `__END__` tokens, which must be on a line by themselves.

The profile is named by placing a name following a `__NAME__` pragma:

```
__NAME__ billing
```

The `__NAME__` must begin the line, and be followed by whitespace and the name. The search profile can then be accessed by <mv_order_profile="billing">. See Advanced Multi−level Order Pages.

26.49. OrderReport

The location of the simple order report file. Defaults to etc/report.

```
OrderReport          /data/order−form
```

26.50. PageDir

Location of catalog pages. Defaults to the pages subdirectory in the VendRoot directory.

```
PageDir    /data/catalog/pages
```

Can be set in the `Locale` settings to allow different page sets for different locales.

26.51. PageSelectField

Sets a products database column which can be used to select the on−the−fly template page. This allows multiple on−the−fly pages to be defined. If the field is empty (no spaces), the default `flypage` will be used.

```
PageSelectField    display_page
```

26.52. ParseVariables

Determines whether global and catalog variables will be parsed in the configuration file. Default is No. The foundation catalog.cfg turns ParseVariables on and usually expects it to be on.

```
Variable STORE_ID  topshop
ParseVariables Yes
StaticDir  /home/__STORE_ID__/www/cat
ParseVariables No
```

26.53. Password

The encrypted or unencrypted password (depending on Variable MV_NO_CRYPT) that will cause internal authorization checks for RemoteUser to allow access.

Below is the encrypted setting for a `blank` password.

```
Password                bAWoVkuzphOX.
```

26.54. PGP

If credit card information is to be accepted, and the e−mailed order will go over an insecure network to reach its destination, PGP security should be used. The key ring to be used must be for the user that is running the Interchange server, or defined by the environment variable `PGPPATH`, and the key user specified must have a key on the public key ring of that user.

```
PGP        /usr/local/bin/pgp −feat orders@company.com
```

If this directive is non−null, the PGP command string as specified will be used to encrypt the entire order in addition to any encryption done as a result if `CreditCardAuto`. If, for some reason, an error comes from PGP, the customer will be given the special page `failed`.

If a `Route` is set up to `supplant`, this is ignored.

26.55. Pragma

Sets the default value of an Interchange pragma. The directive is set like this:

```
Pragma my_pragma_name
```

To enable a pragma for only a particular page, set it anywhere in the page:

```
[pragma my_pragma_name]
```

To disable a pragma for a particular page, set it anywhere in the page:

```
[pragma my_pragma_name 0]
```

Descriptions of each pragma follow.

dynamic_variables

dynamic_variables_file_only

no_html_parse

Disallows HTML tag parsing. This is a **big** parser performance gain and is enabled in the demo catalog.

When this pragma is set, you can't encapsulate Interchange tags inside HTML tags like this:

```
<P MV="if scratch something"> ... </P>
```

Note that a page with no HTML parsing is a good place to put a DTD (document type descriptor).

no_image_rewrite

Prevents image locations in pages from being altered by Interchange. Added in Interchange 4.7.0.

Interchange normally rewrites image locations to point to ImageDir. This applies to image locations mentioned in <img src="...">, <input src="...">, <body background="...">, <table background="...">, and <tr/th/td background="...">.

When this pragma is **not** set, the following tag:

```
<img src="fancy.gif">
```

Would, assuming an ImageDir set to /foundation/images, be transformed into:

```
<img src="/foundation/images/fancy.gif">
```

When pragma no_image_rewrite **is** set, the <img> tag would remain unchanged.

safe_data

By default Interchange does not allow data returned from databases to be reparsed for Interchange tags. Setting the safe_data pragma eliminates this restriction.

If for some reason you want to have tags in your database, for example, to use [page ...] for catalog−internal hyperlinks in your product descriptions, you need to enable safe_data. Some things to consider:

1. It may be better to use the safe_data attribute available to certain tags instead of the pragma, or perhaps to use [pragma] for a whole page or [tag pragma] ... [/tag] for a small block, instead of a catalog−wide Pragma directive.
2. In any case it is strongly recommended that you surround the area with [restrict] ... [/restrict] tags to allow only the specific (hopefully relatively safe) set of tags you expect to appear, such as [page] or [area]. Expect security compromises if you allow [calc] or [perl], or other extremely powerful tags.
3. Be certain that you know everywhere the data in your database will be used. Will it always be possible to reparse for tags? What about when it's used to create an emailed plain−text receipt −− will a literal '[page ...]' tag show up in the product description on the receipt? Would the desired output of '<a href="...">' be any better in a plaintext situation? What if you access your database from applications other than Interchange? You'll then have to decide what to do with such tags; perhaps you can simply strip them, but will the missing tag output cause you any trouble?

In short, safe_data is disabled by default for a reason, and you should be very careful if you decide to enable it.

(Watch out for parse order with [tag pragma] or [restrict] when used with lists that retrieve data from the database, as in [prefix−*] and the flypage. Loops parse before regular tags like [tag] and [restrict], and thus aren't affected by it.)

strip_white

Set this to strip whitespace from the tops of HTML pages output by Interchange. Such whitespace usually comes from Interchange tags at the top of the page. The pragma's purpose is mostly to make 'view source' in the browser a slightly more tolerable experience.

Default is off; whitespace is unchanged.

26.56. PriceCommas

If no commas are desired in price numbers (for the `[item−price]` tag), set this to `No`. The default is to use commas (or whatever is the thousands separator for a locale).

```
PriceCommas         no
```

This is overridden if a Locale `price_picture` is set.

26.57. PriceDivide

The number the price should be divided by to get the price in units (dollars or such). The default is one. If penny pricing is used, set it to 100.

```
PriceDivide         100
```

Can be set in the `Locale` settings to allow a price adjustment factor for different currencies.

26.58. PriceField

The field in the product database that will be accessed with the `[item−price]` element. Default is "price."

```
PriceField          ProductPrice
```

Can be set in the `Locale` settings to allow different price fields for different currencies.

26.59. ProductDir

Location of the database files. Defaults to the products subdirectory of the VendRoot directory. May not be set to an absolute directory unless `NoAbsolute` is defined as No.

```
ProductDir          /data/catalog/for−sale
```

Most people never set this directive and use the default of `products`.

26.60. ProductFiles

Database tables that should be seen as the "products" database.

```
ProductFiles    vendor_a vendor_b
```

The key thing about this is that each will be searched in sequence for a product code to order or an `[item-field ....]` or `[loop-field ...]` to insert. The main difference between `[item-field ....]` and `[item-data table ...]` is this fall-through behavior.

Default is `products`.

26.61. ReadPermission and WritePermission

By default, only the user account that Interchange runs under (as set by the SETUID permission on vlink) can read and write files created by Interchange. `WritePermission` and `ReadPermission` can be set to `user`, `group`, or 'world'.

```
ReadPermission      group
WritePermission     group
```

26.62. RemoteUser

The value of the HTTP environment variable `REMOTE_USER` that will enable catalog reconfiguration. HTTP basic authentication must be enabled for this to work. Default is blank, disabling this check.

```
RemoteUser   interchange
```

26.63. Replace

Causes a directive to be emptied and re-set (to its default if no value is specified). Useful for directives that add to the value by default.

```
Replace NoCache ord special multi reconfig query
```

Capitalization must be exact on each directive.

26.64. Require

Forces a Perl module, global `UserTag`, or `GlobalSub` to be present before the catalog will configure. This is useful when transporting catalogs to make sure they will have all needed facilities.

```
Require usertag   email
Require globalsub form_mail
Require module    Business::UPS
```

26.65. RobotLimit

The RobotLimit directive defines the number of consecutive pages a user session may access without a 30 second pause. If the limit is exceeded, the command defined in the Global directive `LockoutCommand` will be executed and catalog URLs will be rewritten with host 127.0.0.1, sending the robot back to itself. The default is 0, disabling the check.

```
RobotLimit  200
```

26.66. Route

Sets up order routes. See `Custom Order Routing`. There are examples in the demo `simple`.

26.67. SalesTax

If non−blank, enables automatic addition of sales tax based on the order form. The value is a comma−separated list of the field names (as placed in order.html) in priority order, which should be used to look up sales tax percentage in the `salestax.asc` database. This database is not supplied with Interchange. It is typically received from a third party by quarterly or monthly subscription.

```
SalesTax            zip state
```

26.68. SalesTaxFunction

A Perl subroutine that will return a hash reference with the sales tax settings. This can be used to query a database for the tax for a particular vendor:

```
SalesTaxFunction  <<EOR
      my $vendor_id = $Session->{source};
    my $tax = $TextSearch->hash( {
                        se => $vendor_id,
                        fi => 'salestax.asc',
                        sf => 'vendor_code',
                        ml => 1000,
                        } );
    $tax = {} if ! $tax;
    $tax->{DEFAULT} = 0.0;
    return $tax;
EOR
```

or simply produce a table:

```
SalesTaxFunction  <<EOR
 return {
     DEFAULT => 0.0,
     IL => 0.075,
     OH => 0.065,
 };
EOR
```

A `DEFAULT` value must always be returned or the function will be ignored.

26.69. SaveExpire

The default amount of time that a cookie will be valid (other than the MV_SESSION_ID cookie). The ones used in Interchange by default are `MV_USERNAME` and `MV_PASSWORD` for the `CookieLogin` feature. Specified the same as `SessionExpire`, with an integer number followed by one of `minutes`, `hours`, `days`, or `weeks`.

```
SaveExpire 52 weeks
```

Default is `30 days`.

26.70. ScratchDefault

The default scratch variable settings that the user will start with when their session is initialized. To disable placing URL rewrite strings after the user has given a cookie, set:

```
ScratchDefault  mv_no_session_id  1
ScratchDefault  mv_no_count       1
ScratchDefault  mv_add_dot_html   1
```

26.71. ScratchDir

The directory where temporary files will be written, notably cached searches and retired session IDs. Defaults to `tmp` in the catalog directory.

```
ScratchDir          /tmp
```

26.72. SearchProfile

Allows an unlimited number of search profiles to be set up, specifying complex searches based on a single click. The directive accepts a file name based in the catalog directory if the path is relative:

```
SearchProfile    etc/search.profiles
```

As an added measure of control, the specification is evaluated with the special Interchange tag syntax to provide conditional setting of search parameters. The following file specifies a dictionary−based search in the file 'dict.product':

```
__NAME__ dict_search
mv_search_file=dict.product
mv_return_fields=1
[if value fast_search]
  mv_dict_limit=-1
  mv_last=1
[/if]
__END__
```

The `__NAME__` is the value to be specified in the `mv_profile` variable on the search form, as in

```
<INPUT TYPE=hidden NAME=mv_profile VALUE="dict_search">
```

or with mp=profile in the one−click search.

```
[page scan se=Renaissance/mp=dict_search]Renaissance Art[/page]
```

Multiple profiles can reside in the same file, if separated by __END__ tokens. __NAME__ tokens should be left−aligned, and __END__ must be on a line by itself with no leading or trailing whitespace.

26.73. SecureURL

The base URL for secure forms/page transmissions. Normally it is the same as VendURL except for the `https:` protocol definition. Default is blank, disabling secure access.

```
SecureURL   https://machine.com/xyzcorp/cgi−bin/vlink
```

26.74. SendMailProgram

The location of the sendmail binary, needed for mailing orders. Must be found at startup. This often needs to be set for FreeBSD or BSDI.

```
SendMailProgram    /usr/sbin/sendmail
```

If set to `none`, no mail can be sent by standard Interchange facilities. The default is the value in `interchange.cfg` and varies depending on operating system.

26.75. SeparateItems

Changes the default when ordering an item via Interchange to allowing multiple lines on the order form for each item. The default, `No`, puts all orders with the same part number on the same line.

Setting `SeparateItems` to `Yes` allows the item attributes to be easily set for different instances of the same part number, allowing easy setting of things such as size or color.

```
SeparateItems       Yes
```

Can be overridden with the `mv_separate_items` variables (both scratch and values).

26.76. SessionDatabase

When storing sessions, specify the name of the directory or DBM file to use. The file extensions of .db or .gdbm (depending on the DBM implementation used) will be appended. If the default file−based sessions are used, it is the name of the directory.

```
SessionDatabase     session−data
```

Can be an absolute path name, if desired.

It is possible for multiple catalogs to share the same session file, as well as for multiple Interchange servers to serve the same catalogs. If serving a extremely busy store, multiple parallel Interchange servers can share the same NFS−based file system and serve users in a "ping−pong" fashion using the file−based sessions. On huge

systems, the level of directory hashing may be changed. By default, only 48 * 48 hashing is done. See the source for `SessionFile.pm`.

26.77. SessionDB

The name of the Interchange database to be used for sessions if DBI is specified as the session type. This is not recommended.

26.78. SessionExpire

A customer can exit the browser or leave the catalog pages at any time, and no indication is given to the web server aside from the lack of further requests that have the same session ID. Old session information needs to be periodically expired. The `SessionExpire` specifies the minimum time to keep track of session information. Defaults to one day. Format is an integer number, followed by s(econds), m(inutes), h(ours), d(ays), or w(eeks).

```
SessionExpire       20 minutes
```

If `CookieLogin` is in use, this can be a small value. If the customer's browser has the Interchange session cookie stored, he/she will be automatically logged back in with the next request. Note, however, that the customer's cart and session values will be reset.

26.79. SessionLockFile

The file to use for locking coordination of the sessions.

```
SessionLockFile     session-data.lock
```

This only applies when using DBM-based sessions. It is possible for multiple catalogs to share the same session file. `SessionDatabase` needs to be set appropriately if the database is to be shared. Defaults to `session.lock`, which is appropriate for separate session files (and therefore standalone catalogs). Can be an absolute path name, if desired.

26.80. SessionType

The type of session management to be used. Use one of the following:

```
DB_File     Berkeley DB
DBI         DBI (don't use this, normally)
File        File-based sessions (the default)
NFS         File-based sessions, forces use of fcntl locking
GDBM        GDBM
```

The default is file-based sessions, which provides the best performance and reliablility in most environments.

If you are planning on running Interchange servers with an NFS-mounted filesystem as the session target, you must set SessionType to "NFS". The other requisites are usually:

1. fcntl() supported in Perl 2. lock daemon running on NFS server system 3. lock daemon running on Interchange server

See also the global directive LockType.

26.81. SpecialPage

Sets a special page to other than its default value. Can be set as many times as necessary. Will have no effect if not one of the Interchange Required Pages.

```
SpecialPage         checkout ord/checkout
SpecialPage         failed special/error_on_order
SpecialPage         interact special/browser_problem
SpecialPage         noproduct special/no_product_found
SpecialPage         order  ord/basket
SpecialPage         search srch/results
```

26.82. SpecialPageDir

The directory where special pages are kept. Defaults to `special_pages` in the catalog directory.

```
SpecialPageDir      pages/special
```

26.83. Static

A Yes/No directive. Enables static page building and display features. Default is `No`.

```
Static   Yes
```

26.84. StaticAll

A Yes/No directive. Tells Interchange to try and build all pages in the catalog statically when called with the static page build option. This is subject to the settings of `StaticFly`, `StaticPath`, and `NoCache`. Default is No. Pages that have dynamic elements will not be built statically, though that may be overridden with `[tag flag build][/tag]` on the page in question.

```
StaticAll   Yes
```

26.85. StaticDepth

The number of levels of static search building that will be done if a search results page contains a search. Default is one, though it could be very long if set higher. Set to 0 to disable re−scanning of search results pages.

```
StaticDepth 2
```

26.86. StaticDir

The absolute path of the directory which should be used as the root for static pages. The user ID executing Interchange must have write permission on the directory (and all files within) if this is to work.

```
StaticDir   /home/you/www/catalog
```

26.87. StaticFly

A Yes/No directive. If set to `Yes`, static builds will attempt to generate a page for every part number in the database using the on−the−fly page build capability. If pages are already present with those names, they will be overwritten. The default is `No`.

```
StaticFly   Yes
```

26.88. StaticPage

Tells Interchange to build the named page (or pages, whitespace separated) when employing the static page−building capability of Interchange. Not necessary if using `StaticAll`.

```
StaticPage   info/about_us  info/terms_and_conditions
```

26.89. StaticPath

The path (relative to HTTP document root) which should be used in pages built with the static page−building capability of Interchange.

```
StaticPath    /catalog
```

26.90. StaticPattern

A perl regular expression which is used to qualify pages that are to be built statically. The default is blank, which means all pages qualify.

```
StaticPattern  ^info|^help
```

26.91. StaticSuffix

The extension to be appended to a normal Interchange page name when building statically. Default is `.html`. Also affects the name of pages in the Interchange page directory. If set `to .htm`, the pages must be named with that extension.

```
StaticSuffix   .htm
```

26.92. Sub

Defines a catalog subroutine for use by the `[perl][/perl]` or [mvasp] embedded perl languages. Use the "here document" capability of Interchange configuration files to make it easy to define:

```
Sub <<EOF
sub sort_cart_by_quantity {
    my($items) = @_;
    $items = $Items if ! $items;
    my $out = '<TABLE BORDER=1>';
    @$items = sort { $a->{quantity} <=> $b->{quantity} } @$items;
    foreach $item (@$items) {
        my $code = $item->{code};
```

```
            $out .= '<TR><TD>';
            $out .= $code;
            $out .= '</TD><TD>';
            $out .= $Tag->data('products', 'name', $code);
            $out .= '</TD><TD>';
            $out .= $Tag->data('products', 'price', $code);
            $out .= '</TD></TR>';
        }
        $out .= '&lt/TABLE>';
        return $out;
    }
    EOF
```

As with Perl "here documents," the EOF (or other end marker) must be the ONLY thing on the line, with no leading or trailing white space. Do not append a semicolon to the marker. The above would be called with:

```
[perl]
    my $cart = $Carts->{main};
    return sort_cart($cart);
[/perl]
```

and will display an HTML table of the items in the current shopping cart, sorted by the quantity. Syntax errors will be reported at catalog startup time.

Catalog subroutines may not perform unsafe operations. The Safe.pm module enforces this unless global operations are allowed for the catalog. See `AllowGlobal`.

26.93. Suggests

Generates a warning message when a Perl module, global `UserTag`, or `GlobalSub` is not present at catalog configuration time. Same as the `Require` directive except not fatal.

```
Suggest usertag   table_editor
Suggest globalsub file_info
Suggest module    Business::UPS
```

26.94. TableRestrict

Used to provide "views" in database−based searches. Does not affect the text searches. Affects the table being searched.

Takes the form of `field=session_param`, where `field` is a column in the table being iterated over, and `session_param` is a `$Session` key (i.e., [data session username]).

```
TableRestrict  products  owner=username
```

The above would prevent the database search from returning any records except those where the column `owner` contains the current value of `[data session username]`.

Probably most usefully set by embedded Perl code in certain situations. For example:

```
[calc]
    # Restrict edit to owned fields
    $Config->{TableRestrict}{products} = 'owner=username';
```

```
    return;
[/calc]
```

When using SQL−based databases, in effect it turns the base search query

```
select * from products
```

into

```
select * from products where owner = '[data session username]'
```

Interchange databases are similarly affected, though the methodology is different. Also may be useful in "mall" situations, where user is allowed to only see products from the current store ID.

26.95. TaxShipping

A comma or space−separated list of states or jurisdictions that tax shipping cost, i.e., UT. Blank by default, never taxing shipping.

```
TaxShipping         UT,NV,94024
```

26.96. TrackFile

Name of a logfile that tracks user traffic. This is used in the back office administration report on traffic by affiliate.

The default is that no such file is kept.

26.97. UpsZoneFile

The file containing the UPS zone information, specified relative to the catalog directory unless it begins with a /. It can be in the format distributed by UPS or can be in a tab−delimited format, with the three−letter zip prefix of the customer used to determine the zone. It interpolates based on the value in `mv_shipmode`. A user database named the same as the mv_shipmode variable must be present or the lookup will return zero.

IMPORTANT NOTE: Zone information and updated pricing from UPS must be obtained in order for this to work properly. The zone information is specific to a region!

```
UpsZoneFile         /usr/interchange/data/ups_zone.asc
```

26.98. UseModifier

Determines whether any attributes, the modifiers specified in the directive, can be attached to the item. See `Item Attributes`. The default is no modifier. Don't use a value of `quantity` or this directive will not work properly.

```
UseModifier         size,color
```

Some values are used by Interchange and are not legal:

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

26.99. ValuesDefault

Sets the initial state of the user values, i.e., [value key] or $Values−>{key}.

```
ValuesDefault   fname  New
ValuesDefault   lname  User
```

When the user session starts, `[value fname] [value lname]` will be "New User."

26.100. Variable

Defines a catalog variable that will be available in the current catalog with the notation `__Variable__`. Variable identifiers must begin with a capital letter, and can contain only word characters (**A−Z,a−z,0−9** and underscore). These are substituted second (right after global Variables) in any Interchange page, and can contain any valid Interchange tags except global variables.

```
Variable   DOCUMENT_ROOT   /usr/local/etc/httpd/htdocs
```

26.101. VariableDatabase

The name of a database containing a field Variable which will be used to set Interchange variable values. For example, a database defined as:

```
Database  var var.txt TAB
VariableDatabase var
```

and containing

```
code    Variable
HELLO   Hi!
```

would cause `__HELLO__` to appear as `Hi!`.

The field name is case−sensitive, and `variable` would not work.

The values are inserted at time of definition. Any single−level hash−oriented Interchange directive, such as `SpecialPage`, `ScratchDefault`, or `ValuesDefault`, can be set in the same way. If the `VariableDatabase` named does not exist at definition time, a database of the default type with an ASCII file source appending `.txt` is assumed. In other words:

```
VariableDatabase variable
```

is equivalent to

```
Database         variable variable.txt TAB
VariableDatabase variable
```

26.102. VendURL

Specifies the base URL that will run vlink as a cgi−bin program.

```
VendURL  http://machine.company.com/cgi-bin/vlink
```

26.103. WideOpen

Disables IP qualification of user sessions. **This degrades catalog security.** Do not use unless using encryption or a real−time payment gateway. line:

Foundation Store

27. The Foundation Store

The Foundation store is distributed with Interchange to give you a starting point with which to build your e−business. While the Foundation store is designed to be relatively easy to start with, it is still a full−featured demonstration of a number of Interchange capabilities. Once you understand the Foundation store and how it works you are well on your way to understanding the Interchange software.

The following is a list of some popular features:

Category Searches

Regardless of the number of products in a catalog, categorizing them makes them easier to find. Pick a field in the database, typically named `category`, and classify the products for search using Interchange.

Images

You can display a thumbnail image for the items that have images. To do this, add an image field in the database. (See the 'image' field of the products database.)

Related Items

You can embed searches of similar products on an individual product display page with the `[query ...]` or `[loop ...]` tags. Or, if customer data is developed, search a past order database and display products that would be of interest to that customer.

Reviews/Testimonials

You can key the placement of a review or testimonial on the existence of a file being in a certain directory. This is reasonable to do when a user is viewing a single product.

28. Tree design

By determining how users will enter and exit the catalog, complex and intelligent conditional schemes are possible, especially if the Cookies capability is exercised. However, it is recommended that simplicity be used. Consumers will not make purchases if they can't navigate their way around the catalog.

It is important to remember that users will lose their session (and items in their shopping cart) if their browser does not accept cookies and they leave the site. Interchange addresses this problem by using the `area` and `page` tags. If you are using frames, source all frame panes containing Interchange links from an initial page served by Interchange. If you don't do this, the user may have multiple session IDs depending on which frame generated the link.

Note that Interchange can work properly even if the browser doesn't store cookies. In this situation Interchange inserts a session ID into each URL; if the ID is preserved as the user navigates from page to page the session will remain intact.

29. The Catalog Directory

Interchange pages are contained in the catalog directory. Each individual catalog has its own base directory. The catalog directory has the following structure by default:

catalog.cfg

File containing configuration directives for a particular catalog. Configuration settings established in the catalog.cfg directory will not effect any other catalogs running under the version of Interchange you are using. Subcatalogs can have differing information in a file named for that subcatalog.

config

Directory that will be read when directives are set with the `filename` notation. For example, the file `config/static.pages` will be read when the following directive is encountered in the `catalog.cfg` file.

```
StaticPage  <static.pages
```

This directory also contains template information used with the `makecat` program.

error.log

File which contains catalog−specific errors. It is also where any syntax errors in embedded Perl code are shown.

etc

Directory normally used for tracking files, order profiles, and other configuration and log information.

pages

Directory that contains the pages of the catalog. This can be considered to be the "document root" of the catalog. Pages contained therein are called with the path information after the script name. For example:

```
/cgi−bin/simple/products/gold will call the page in the file
pages/products/gold.html.
```

products

Directory that contains database source files, including the special Interchange databases shipping.asc, pricing.asc (and other shipping database files).

session

Directory that contains session files.

tmp

The temporary or scratch directory used for various storage reasons, like retired ID numbers, search paging

files, sort tests, import temporary files, etc. This is the default set by ScratchDir. It can be redefined to be located on another partition.

30. Page Templates

This section describes the files located in the Foundation demo.

30.1. Template File Locations

This diagram shows the directory and file structure used for the default Foundation 'templates' directory. The base will be a directory with the name of your catalog, here called CATROOT.

```
CATROOT/
|
|----templates/
     |----cart
     |----components/
          |----affiliate_receptor
          |----best_horizontal
          |----best_vertical
          |----cart
          |----cart_display
          |----cart_tiny
          |----category_vertical
          |----cross_horizontal
          |----cross_vertical
          |----modular_buy
          |----modular_update
          |----none
          |----promo
          |----promo_horizontal
          |----promo_vertical
          |----random
          |----random_horizontal
          |----random_vertical
          |----saved_carts_list_small
          |----search_box_small
          |----upsell
          |----upsell_horizontal
          |----upsell_vertical
     |----default --> foundation
     |----foundation/
          |----cart
          |----fullwidth
          |----leftonly
          |----leftright
          |----regions/
               |----LEFTONLY_BOTTOM
               |----LEFTONLY_TOP
               |----LEFTRIGHT_BOTTOM
               |----LEFTRIGHT_TOP
               |----NOLEFT_BOTTOM
               |----NOLEFT_TOP
          |----simple
          |----theme.cfg
     |----fullwidth
     |----leftonly
     |----leftright
     |----regions/
     |----LEFTONLY_BOTTOM
     |----LEFTONLY_TOP
```

```
|----LEFTRIGHT_BOTTOM
|----LEFTRIGHT_TOP
|----NOLEFT_BOTTOM
|----NOLEFT_TOP
|----sampledata/
|----computers/
     |----images/
          |----items/
               |----generic.gif
               |----gift_certificate_large.gif
               |----yourimage.gif
          |----thumb/
               |----generic_thumb.gif
               |----gift_certificate.gif
               |----thumb.gif
     |----products/
          |----inventory.txt
          |----merchandising.txt
          |----mv_metadata.asc
          |----options.txt
          |----pricing.txt
          |----products.txt
          |----userdb.txt
|----reports/
     |----download/
          |----00352as.pdf
          |----11993ab.pdf
          |----22083da.pdf
          |----49503cg.pdf
          |----59330rt.pdf
          |----59402fw.pdf
          |----66548ch.pdf
          |----73358ee.pdf
          |----83491vp.pdf
          |----90773sh.pdf
     |----products/
          |----mv_metadata.asc
          |----products.txt
          |----userdb.txt
|----tools/
     |----etc/
          |----after.cfg
          |----before.cfg
     |----images/
          |----items/
               |----os28004.gif
               |----os28005.gif
               |----os28006.gif
               |----os28007.gif
               |----os28008.gif
               |----os28009.gif
               |----os28011.gif
               |----os28044.gif
               |----os28057a.gif
               |----os28057b.gif
               |----os28057c.gif
               |----os28062.gif
               |----os28064.gif
               |----os28065.gif
               |----os28066.gif
               |----os28068.gif
               |----os28068a.gif
```

```
        |----os28068b.gif
        |----os28069.gif
        |----os28070.gif
        |----os28072.gif
        |----os28073.gif
        |----os28074.gif
        |----os28075.gif
        |----os28076.gif
        |----os28077.gif
        |----os28080.gif
        |----os28081.gif
        |----os28082.gif
        |----os28084.gif
        |----os28085.gif
        |----os28086.gif
        |----os28087.gif
        |----os28108.gif
        |----os28109.gif
        |----os28110.gif
        |----os28111.gif
        |----os28112.gif
        |----os28113.gif
        |----os29000.gif
    |----thumb/
        |----gift_certificate.gif
        |----os28004_b.gif
        |----os28005_b.gif
        |----os28006_b.gif
        |----os28007_b.gif
        |----os28008_b.gif
        |----os28009_b.gif
        |----os28011_b.gif
        |----os28044_b.gif
        |----os28057a_b.gif
        |----os28057b_b.gif
        |----os28057c_b.gif
        |----os28062_b.gif
        |----os28064_b.gif
        |----os28065_b.gif
        |----os28066_b.gif
        |----os28068_b.gif
        |----os28068a_b.gif
        |----os28068b_b.gif
        |----os28069_b.gif
        |----os28070_b.gif
        |----os28072_b.gif
        |----os28073_b.gif
        |----os28074_b.gif
        |----os28075_b.gif
        |----os28076_b.gif
        |----os28077_b.gif
        |----os28080_b.gif
        |----os28081_b.gif
        |----os28082_b.gif
        |----os28084_b.gif
        |----os28085_b.gif
        |----os28086_b.gif
        |----os28087_b.gif
        |----os28108_b.gif
        |----os28109_b.gif
        |----os28110_b.gif
        |----os28111_b.gif
```

```
            |----os28112_b.gif
            |----os28113_b.gif
            |----os29000_b.gif
       |----products/
            |----affiliate.txt
            |----area.txt
            |----cat.txt
            |----inventory.txt
            |----merchandising.txt
            |----mv_metadata.asc
            |----options.txt
            |----orderline.txt
            |----pricing.txt
            |----products.txt
            |----transactions.txt
            |----userdb.txt
```

30.2. Themes

This section explains how themes are defined in Interchange via the STYLE variable and the theme configuration file, theme.cfg.

30.2.1. STYLE

The STYLE variable in CATROOT/products/variable.txt indicates the template style to be used as the theme for the catalog; the appropriate templates for that theme are found in CATROOT/templates/__STYLE__/. (To change the value of the STYLE variable, either edit variable.txt directly or use the table editor feature of the admin interface.)

The default theme for Interchange is the Foundation demo; hence, the STYLE variable is assigned the value 'Foundation' in variable.txt. The theme is defined in catalog.cfg as follows (line numbers added):

```
    # Here we set up the catalog theme.

1 ParseVariables Yes

2 ifndef STYLE
3 Variable STYLE default
4 endif
5 include templates/__STYLE__/theme.cfg
```

Variables that make up the look and feel of the STYLE (theme) are defined in the file CATROOT/templates/foundation/theme.cfg, which is read by Interchange in line 5 above.

30.2.2. theme.cfg

The file CATROOT/templates/foundation/theme.cfg serves three purposes:

1. It defines the THEME and THEME_IMG_DIR variables,
2. It defines a cascading style sheet for the theme, and
3. It defines the location of region templates according to the traffic settings for the Interchange daemon.

The THEME variable is used to set the location of the region templates in the traffic settings section of the theme.cfg file. It is also used in the cart template definition file (CATROOT/templates/cart) to set the path of

an image. The THEME_IMG_DIR variable is used to set image paths in the template region files and the template component files.

The look and feel of the Foundation theme are defined primarily in the cascading style sheet specified in the theme.cfg file. This

The Interchange TRAFFIC setting, defined system−wide in interchange.cfg, is described in the ??document??. The settings in theme.cfg pertain to the location of region templates for the high and low traffic settings. For example, if you need to define a separate set of high traffic templates, you would change the ConfigDir variable in theme.cfg to point to the directory containing those templates.

30.3. Template Definition Files

The template definition files store the name and description of the template as well as components and options for that template.

```
templates/cart
templates/fullwidth
templates/leftonly
templates/leftright

templates/foundation/cart
templates/foundation/fullwidth
templates/foundation/leftonly
templates/foundation/leftright
templates/foundation/simple
```

30.3.1. Template Walkthrough −− leftonly

This section is best read while viewing the file CATROOT/templates/leftonly and the 'Edit Page' page in the Content Editor of the Interchange Administration Tool.

Looking at the example template definition file, all lines located between the [comment] and [/comment] tags (lines 1 and 53) control what is available in the Edit Page screen of the Administration Tool.

Lines 2−5: Template specification

```
2  ui_template: Yes
3  ui_template_name: leftonly
4  ui_template_layout: LEFTONLY_TOP, UI_CONTENT, LEFTONLY_BOTTOM
5  ui_template_description: Page with top/left areas.
```

Line 2 indicates that this file is a template for the user interface. Line 3 names the template, while Line 4 indicates the regions that comprise the template and that will eventually make up the new page that is created from the template. Line 5 provides a description used to identify the template when it appears in a Select Template pull−down menu on the Edit Page of the Administration Tool. This description can be changed or modified to better describe a new template or a template that is created from the stock templates provided with Interchange.

Lines 7−8: Break

```
7  break:
8          widget: break
```

This code creates a separation line in the Edit Page between sets of options. In the default Interchange installation the line is grey, but the color can be changed. Note −− Changing this color applies the change to any catalog served by Interchange.

Lines 10−11: Page Title

```
10 page_title:
11         description: Page title
```

This code tells Interchange to display a text field on the Edit Page for entering the page title ('Title of New Page' in this example). The value entered is assigned to the scratch variable page_title and is set as a default value at the bottom of the template definition file using the following ITL:

```
54 [set page_title][set]
```

which, in turn, sets the scratch variable on the new page using the tag

```
[set page_title]Title of New Page[set]
```

The scratch variable page_title is parsed by the following code in the region template specified in the template definition file and called in the new page:

```
   <title>[scratch page_title]</title>
```

Lines 13−15: Page Banner

```
13 page_banner:
14         description: Page banner
15         help: Defaults to page title
```

Assigns a textual title for the page to the scratch variable page_banner, which is assigned by the following ITL:

```
55 [set page_banner][set]
```

The scratch variable page_banner is set on the new page using the tag

```
[set page_banner]Banner of New Page[set]
```

The scratch variable can be parsed in the region template by this code:

```
[either]
    [scratch page_banner]
[or]
    [scratch page_title]
[/either]
```

This results in the page banner being displayed if defined. Otherwise, the page title is used.

Lines 17−20: Members Only

```
17 members_only:
18         options: 1=Yes,0=No*
19         widget: radio
```

```
20         description: Members only
```

This creates a radio−button form element on the Edit Page with the user can specify whether the page can be accessed if a visitor is logged in (Yes) or not (No). The default is indicated by an asterisk.

The scratch variable members_only is assigned by the ITL code

```
56 [set members_only][set]
```

and set on the new page using the tag

```
   [set members_only]0[/set]
```

if the page can be accessed without logging in or

```
   [set members_only]1[/set]
```

if it can not.

The members_only function is handled by the following code within each region template file:

```
[if scratch members_only]
  [set members_only][/set]
  [if !session logged_in]
    [set mv_successpage]@@MV_PAGE@@[/set]
    [bounce page=login]
  [/if]
[/if]
```

This code says that if "members only" is set to yes, and the visitor is logged in, to display the page. Otherwise, redirect the visitor to the login page.

Lines 22−23: Break

```
22 break1:
23         widget: break
```

Another separation line.

Lines 25−28: Horizontal Before Component

```
25 component_before:
26         options: =none, best_horizontal=Best Sellers, cross_horizontal=Cross sell, \
            promo_horizontal=Promotion, random_horizontal=Random items, \
            upsell_horizontal=Upsell
27         widget: select
28         description: Component before content
```

This allows the inclusion of a defined component (included in the CATROOT/templates/components directory) to be displayed before, or above, the page's content. It provides a pull−down menu on the Edit Page displaying the available components. The components, identified here on line 26, can be assigned a name via the value=name convention.

The scratch variable component_before is assigned in the template definition file by the ITL code

```
57 [set component_before][set]
```

It is called with the following code within the LEFTRIGHT_TOP, LEFTONLY_TOP, and NOLEFT_TOP region templates:

```
[if scratch component_before]
  [include file="templates/components/[scratch component_before]"]
[/if]
```

Lines 30−33: Horizontal After Component

```
30 component_after:
31         options: =none, best_horizontal=Best Sellers, cross_horizontal=Cross sell, \
            promo_horizontal=Promotion, random_horizontal=Random items, \
            upsell_horizontal=Upsell
32         widget: select
33         description: Component after content
```

Similar to component_before, this allows the inclusion of a defined component after, or below, the page's content.

The scratch variable component_before is assigned in the template definition file by the ITL code

```
58 [set component_after][set]
```

It is called with the following code within the LEFTRIGHT_BOTTOM and LEFTONLY_BOTTOM region templates:

```
[if scratch component_after]
  [include file="templates/components/[scratch component_after]"]
[/if]
```

Lines 35−38: Horizontal Item Width

```
35 component_hsize:
36         options: 1,2,3*
37         widget: select
38         description: Component items horizontal
```

This setting allows you to choose how many items the horizontal components display. For example, the horizontal best sellers component ("best_horizontal") uses this setting to randomly select the best sellers. Notice the default is 3 if nothing is defined. It is called by the following code in the promo_horizontal and random_horizontal components in the Foundation demo.

```
random="[either][scratch component_hsize][or]2[/either]"
```

Lines 40−45: Before/After Banner

```
40 hbanner:
41         options: =--custom--, Also see..., Best Sellers, New items, \
            Some of our fine products, Specials, You might also like
42         widget: move_combo
43         width: 40
44         description: Before/after Banner
45         help: Banner for Before/after component
```

Allows a title for the horizontal components to be defined to be displayed in a header above the component's items. It is called with the [scratch hbanner] tag and used in the Foundation demo in the random_horizontal component.

Lines 47−51: Special Tag

```
47 hpromo_type:
48         options: specials=Specials, new=New items
49         widget: select
50         description: Special tag
51         help: Only for a horizontal Promotion
```

This setting is only viable when a promotion is used for a horizontal component. It tells the promotional component which row(s) to evaluate in the merchandising table for display within the component. This setting, used in the promo_horizontal component, typically correlates to the featured column of the merchandising table as follows:

```
[query arrayref=main
       sql="
        SELECT sku,timed_promotion,start_date,finish_date
        FROM merchandising
        WHERE featured = '[scratch hpromo_type]'
        "]
[/query]
```

30.4. Edit Page Function

Creating a page with the following specifications using the Edit Page function results in the HTML and ITL code displayed below.

Specifications:

```
Template:                   Page with top/left areas.
Page title:                 test
Page banner:                test
Members only:               No
Component before content:   Best Sellers
Component after content:    Random items
Component items horizontal: 3
Before/after Banner:        New items
Special tag:                Specials
Content:                    <P>My first HTML/ITL page!
```

Resulting code:

```
[comment]
ui_template: Yes
ui_template_name: leftonly
[/comment]

[set hbanner]New items[/set]
[set page_title]test[/set]
[set hpromo_type]specials[/set]
[set component_hsize]3[/set]
[set page_banner]test[/set]
[set members_only]0[/set]
```

```
[set component_before]best_horizontal[/set]
[set component_after]random_horizontal[/set]
@_LEFTONLY_TOP_@

<!-- BEGIN CONTENT -->
<P>My first HTML/ITL page!
<!-- END CONTENT -->

@_LEFTONLY_BOTTOM_@
```

An important point demonstrated here is the inclusion of the region templates LEFTONLY_TOP and LEFTONLY_BOTTOM through the @_VARIABLE_NAME_@ notation. These are included because of line 4 of the leftonly template definition file:

```
4  ui_template_layout: LEFTONLY_TOP, UI_CONTENT, LEFTONLY_BOTTOM
```

However, understand that you are free to change the region templates used in the file by editing the file itself or, better yet, using an existing region as a starting point for a region of your own design.

The next section explains the structure of region templates.

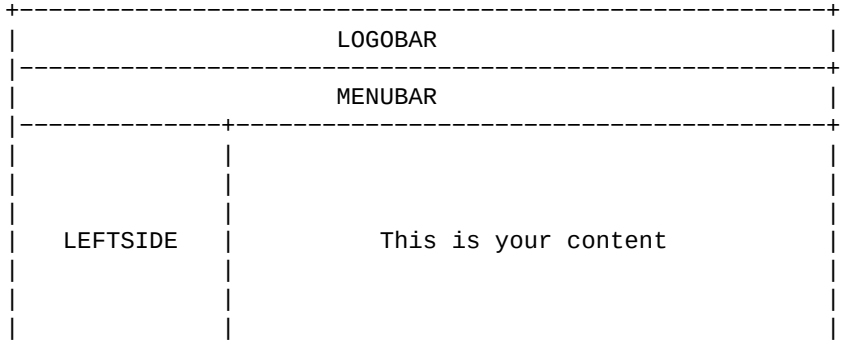
30.5. Region Templates

Interchange region templates (or "regions") are portions of HTML and ITL that are included in pages within a catalog. Using regions, along with the cascading style sheet defined in theme.cfg, allows you to control the look and feel of specific parts of each catalog page.

The default Foundation region set, found in CATROOT/templates/foundation/regions, includes the following:

```
LEFTONLY_TOP
LEFTONLY_BOTTOM
LEFTRIGHT_TOP
LEFTRIGHT_BOTTOM
NOLEFT_TOP
NOLEFT_BOTTOM
```

The Foundation demo uses the Variable feature extensively to simplify hand page editing. Basically, a Variable is a define that permits the substitution of text for a simple __VARIABLE__ string in a page. For example, in the test page above, the variables LEFTONLY_TOP and LEFTONLY_BOTTOM correspond to region templates that provide a logobar, menubar, leftside menu, and copyright footer. Content, consisting of HTML and ITL, is placed within the BEGIN and END CONTENT comments. The following illustration shows how this looks on the page:



```
|              +---------------------------------------|
|              |               COPYRIGHT               |
+-----------------------------------------------------+
```

In this diagram, LEFTONLY_TOP contributes the LEFTSIDE, LOGOBAR, and MENUBAR sections, while LEFTONLY_BOTTOM contributes the COPYRIGHT section.

The following subsections provide an inventory of where each of the region templates, included with the Foundation demo, are used in the pages and template definition files that make up the catalog.

30.5.1. LEFTONLY_TOP

The LEFTONLY_TOP template region is used in the following template pages:

```
pages/aboutus.html
pages/account.html
pages/affiliate/index.html
pages/affiliate/login.html
pages/canceled.html
pages/contact.html
pages/customerservice.html
pages/flypage.html
pages/help.html
pages/login.html
pages/logout.html
pages/modular_modify.html
pages/new_account.html
pages/ord/basket.html
pages/privacypolicy.html
pages/process_return.html
pages/quantity.html
pages/query/check_orders.html
pages/query/order_detail.html
pages/query/order_return.html
pages/returns.html
pages/saved_carts.html
pages/ship_addresses.html
pages/ship_addresses_added.html
pages/ship_addresses_removed.html
pages/stock-alert-added.html
pages/stock-alert.html
```

The LEFTONLY_TOP template region is used in the following templates:

```
templates/foundation/cart
templates/foundation/leftonly
templates/foundation/simple
```

30.5.1.1. Region Template Walkthrough -- LEFTONLY_TOP

```
1  <!-- BEGIN LEFTONLY_TOP -->
2  [if scratch members_only]
3      [set members_only][/set]
4      [if !session logged_in]
5      [set mv_successpage]@@MV_PAGE@@[/set]
6      [bounce page=login]
7      [/if]
8  [/if]
```

```
 9
10  <html>
11  <head>
12    <title>[scratch page_title]</title>
13    __THEME_CSS__
14  </head>
15
16  <body marginheight="0" marginwidth="0">
17
18  <!--- top left and right logo --->
19  <table width="100%" border="0" cellspacing="0" cellpadding="0">
20  <tr>
21    <td align="left" valign="middle" class="maincontent">
22       <img src="__THEME_IMG_DIR____LOGO__">
23    </td>
24    <td align="right" valign="middle" class="maincontent">
25      <img width="174" height="60" src="__THEME_IMG_DIR__logo2.gif"> 
26    </td>
27  </tr>
28  </table>
29
30  <!--- menu bar along the top --->
31  <table width="100%" border="0" cellspacing="0" cellpadding="0">
32  <tr>
33    <td width="100%" class="menubar">
34      <a href="[area index]"><img name="Home" border="0" src="__THEME_IMG_DIR__home.gif"></
35        <img src="__THEME_IMG_DIR__sep.gif">
36      <a href="[area login]">
37      [if session logged_in]
38        <img alt="Log Out" border="0" src="__THEME_IMG_DIR__logout.gif"></a>
39      [else]
40        <img alt="Log In" border="0" src="__THEME_IMG_DIR__login.gif"></a>
41      [/else]
42      [/if]
43        <img src="__THEME_IMG_DIR__sep.gif">
44      <a href="[area ord/basket]"><img alt="Your Cart" border="0" src="__THEME_IMG_DIR__car
45        <img src="__THEME_IMG_DIR__sep.gif">
46      <a href="[area ord/checkout]"><img alt="Check Out" border="0" src="__THEME_IMG_DIR__c
47        <img src="__THEME_IMG_DIR__sep.gif">
48      <a href="[area customerservice]"><img alt="Customer Service" border="0" src="__THEME_
49        <img src="__THEME_IMG_DIR__sep.gif">
50      <a href="[area aboutus]"><img alt="About Us" border="0" src="__THEME_IMG_DIR__about.g
51    </td>
52  </tr>
53  </table>
54
55  <!--- left category column, main content column, and right special column --->
56  <table width="100%" border="0" cellspacing="0" cellpadding="0">
57  <tr>
58    <td width="20%" valign="top" align="left" class="categorybar">
59      <!--Left Sidebar-->
60      <table width="100%" border="0" cellspacing="0" cellpadding="0">
61        [include file="templates/components/[control component none]"][control]
62        [include file="templates/components/[control component none]"][control]
63        [include file="templates/components/[control component none]"][control]
64      </table>
65    </td>
66    <td width="80%" valign="top" align="center" class="maincontent">
67      [include file="templates/components/[control component none]"][control]
68
```

30.5.2. LEFTONLY_BOTTOM

The LEFTONLY_BOTTOM template region is used in the following template pages:

```
pages/aboutus.html
pages/account.html
pages/affiliate/index.html
pages/affiliate/login.html
pages/canceled.html
pages/contact.html
pages/customerservice.html
pages/flypage.html
pages/help.html
pages/login.html
pages/logout.html
pages/modular_modify.html
pages/new_account.html
pages/ord/basket.html
pages/privacypolicy.html
pages/process_return.html
pages/quantity.html
pages/query/check_orders.html
pages/query/order_detail.html
pages/query/order_return.html
pages/returns.html
pages/saved_carts.html
pages/ship_addresses.html
pages/ship_addresses_added.html
pages/ship_addresses_removed.html
pages/stock-alert-added.html
pages/stock-alert.html
```

The LEFTONLY_BOTTOM template region is used in the following templates:

```
templates/foundation/cart
templates/foundation/leftonly
templates/foundation/simple
```

30.5.3. LEFTRIGHT_TOP

The LEFTRIGHT_TOP template region is used in the following template pages:

```
pages/browse.html
pages/index.html
pages/results.html
pages/results_big.html
pages/swap_results.html
```

The LEFTRIGHT_TOP template region is used in the following templates:

```
templates/foundation/leftright
```

30.5.4. LEFTRIGHT_BOTTOM

The LEFTRIGHT_BOTTOM template region is used in the following template pages:

```
pages/browse.html
```

```
pages/index.html
pages/results.html
pages/results_big.html
pages/swap_results.html
```

The LEFTRIGHT_BOTTOM template region is used in the following templates:

```
templates/foundation/leftright
```

30.5.5. NOLEFT_BOTTOM

The NOLEFT_BOTTOM template region is used in the following template pages:

```
pages/ord/checkout.html
pages/splash.html
```

The NOLEFT_BOTTOM template region is used in the following templates:

```
templates/foundation/fullwidth
```

30.5.6. NOLEFT_TOP

The NOLEFT_TOP template region is used in the following template pages:

```
pages/ord/checkout.html
pages/splash.html
```

The NOLEFT_TOP template region is used in the following templates:

```
templates/foundation/fullwidth
```

30.6. Template Page List

/home/ic/catalogs/ft/pages/:

```
aboutus.html
account.html
browse.html
canceled.html
change_password.html
contact.html
customerservice.html
deliver.html
flypage.html
help.html
index.html
login.html
logout.html
lost_password.html
modular_modify.html
new_account.html
privacypolicy.html
process_return.html
quantity.html
results_big.html
```

```
    results_either.html
    results.html
    returns.html
    saved_carts.html
    ship_addresses_added.html
    ship_addresses.html
    ship_addresses_removed.html
    splash.html
    stock-alert-added.html
    stock-alert.html
    swap_results.html
```

/home/ic/catalogs/ft/pages/admin/report_def:

```
    Order%20Status.html
    Products%20to%20edit.html
```

/home/ic/catalogs/ft/pages/admin/reports:

```
    Order%20Status.html
    Products%20to%20edit.html
```

/home/ic/catalogs/ft/pages/affiliate:

```
    index.html
    login.html
```

/home/ic/catalogs/ft/pages/ord:

```
    basket.html
    checkout.html
```

/home/ic/catalogs/ft/pages/query:

```
    check_orders.html
    get_password.html
    order_detail.html
    order_return.html
```

30.7. Special Page List

/home/ic/catalogs/ft/special_pages/:

```
    badsearch.html
    canceled.html
    cc_not_valid.html
    confirmation.html
    failed.html
    interact.html
    missing.html
    needfield.html
    nomatch.html
    noproduct.html
    notfound.html
    order_security.html
    reconfig.html
    sec_faq.html
```

```
security.html
violation.html
```

30.8. Components

- Added new [control] and [control−set] tags to set series of Scratch− like option areas. Used for components in UI content editing.

Interchange components are portions of HTML and ITL that are included in pages within a catalog depending on options set in the Administration Tool. The default component set includes the following:

```
affiliate_receptor
best_horizontal
best_vertical
cart
cart_display
cart_tiny
category_vertical
cross_horizontal
cross_vertical
modular_buy
modular_update
none
promo
promo_horizontal
promo_vertical
random
random_horizontal
random_vertical
saved_carts_list_small
search_box_small
upsell
upsell_horizontal
upsell_vertical
```

/home/ic/catalogs/ft/templates/components:

30.8.1. affiliate_receptor

Not used in Foundation demo

30.8.2. best_horizontal

The best_horizontal component is used in the following templates:

```
templates/foundation/cart
templates/foundation/leftonly
templates/foundation/leftright
```

Not used in Foundation demo pages

30.8.3. best_vertical

The best_vertical component is used in the following template:

```
templates/foundation/leftright
```

Not used in Foundation demo pages

30.8.4. cart

The cart component is used in the following page:

```
pages/ord/basket.html
```

30.8.5. cart_display

The cart_display component creates a small shopping cart that is displayed on the search results page (pages/results.html). It is displayed after an item in a list of results from a search is added to the shopping cart. cart_display is called in results.html by the following include statement:

```
[include file="templates/components/cart_display"]
```

The cart_display component is used in the following pages:

```
pages/results.html
```

30.8.5.1. Component Walkthrough −− cart_display

The remainder of this section is best read in conjunction with the file CATROOT/templates/components/cart_display in a text editor.

Lines 1−6: Component Specification

```
 1  [comment]
 2  ui_component: cart_display
 3  ui_component_group: info
 4  ui_component_label: Smaller cart for display in content area
 5
 6  [/comment]
 7
```

These lines control what is shown in the Edit page screen of the admin interface.

```
 8  <!−− BEGIN COMPONENT [control component cart_display] −−>
```

Line 8 is an HTML comment noting the start of the code for the component. (Note that this can serve as a useful debugging tool to help you locate the component in the resulting HTML generated by Interchange when you view the source of a page loaded in the browser.)

```
 9  [if items]
```

Line 9 checks to see if there are items in the shopping basket. If there are, the remaining code up to the closing [/if] tag on line 64 is executed. If not, Interchanges continues executing the remaining code in results.html (the file that calls the cart_display component).

```
10  <center>
11    <table width="95%" border="0" cellspacing="0" cellpadding="0">
```

```
12      <TR class="contentbar2" VALIGN=TOP>
13        <td align=center class="contentbar2">Action</td>
14        <td class="contentbar2">
15          SKU
16        </td>
17        <td class="contentbar2">
18          Description
19        </td>
20        <td class="contentbar2">
21          Quantity
22        </td>
23        <td class="contentbar2">
24          Price
25        </td>
26        <td class="contentbar2">
27          Extension
28        </td>
29      </TR>
```

Line 10 centers the table started in line 11. Lines 12−29 create a header row in the shopping cart consisting of the header titles Action, SKU, Description, Quantity, Price, and Extension.

```
30      <TBODY>
31  [item-list]
32
```

Line 30 defines the remainder of the table as a section while the [item−list] tag on line 31 tells Interchange to execute the code up to the closing tag ([/item−list] on line 59 for each item the customer has ordered so far.

```
33      <tr class="[item-alternate 2]maincontent[else]contentbar1[/else][/item-alternate]">
34        <td align=center valign=top>
35          [page ord/basket]edit</A>
36        </TD>
37        <td valign=top>[item-code]</TD>
38        <td valign=top>[page [item-code]][item-description]</A>
39        </TD>
40
```

Line 33 begins the next row in the table. The [item−alternate] tag provided as the value of the class attribute tells Interchange to alternate between displaying the rows according to the "maincontent" and "contentbar1" styles (gray and white, respectively).

Lines 34−36 create a link to the shopping cart (basket.html) where the customer can remove or change the quantity of the item ordered.

Line 37 displays the SKU of the item. Lines 38 and 39 provide a link to the product display page (flypage.html) for the item. The [item−description] tag providing the content of the [page] tag enables the item's name to be displayed as the link to the product display page.

```
41  [if-item-modifier gift_cert]
42        <TD ALIGN=CENTER><small>Amount of gift:</small></TD>
43        <TD ALIGN=CENTER>[item-quantity]</TD>
44        <TD ALIGN=right>
45          [item-subtotal]
46        </TD>
47  [else]
48        <TD ALIGN=CENTER>[item-quantity]</TD>
49        <TD ALIGN=right>
```

```
50          [item-price]
51        </TD>
52        <TD ALIGN=right>
53          [item-subtotal]
54        </TD>
55  [/else]
56  [/if-item-modifier]
57      </TR>
58
```

Line 41 checks whether the item is a gift certificate. If it is it displays "Amount of gift:" and the [item-quantity] (number of gift certificates, in this case) under the headings "Quantity" and "Price", respectively. Otherwise, lines 48 through 50 display the quantity and price of the item ordered. Lines 45 or 53 (depending on whether the item is a gift certificate) display the item subtotal (quantity multiplied by price) for the item under the heading "Extension".

```
59  [/item-list]
60  </TBODY>
61  </table>
62  </FORM>
63  </center>
64  [/if]
65
66  <!-- END COMPONENT [control component cart_display] -->
```

Lines 59 through 64 close out the tags for the component, and line 66 indicates the end of the component code.

30.8.6. cart_tiny

The cart_tiny component is used in the following pages:

```
pages/account.html
pages/browse.html
pages/canceled.html
pages/customerservice.html
pages/flypage.html
pages/help.html
pages/index.html
pages/logout.html
pages/modular_modify.html
pages/new_account.html
pages/privacypolicy.html
pages/process_return.html
pages/quantity.html
pages/query/check_orders.html
pages/query/order_detail.html
pages/query/order_return.html
pages/saved_carts.html
pages/ship_addresses.html
```

30.8.7. category_horizontal

Not used in Foundation demo pages or templates.

30.8.8. category_vertical

The category_vertical component provides a listing of all products in the catalog, organized by prod_group (e.g., Hand Tools, Ladders). category_vertical is usually displayed in the LEFTSIDE section of the page, under the search_box_small component.

The category_vertical component is used in the following pages:

```
pages/aboutus.html
pages/account.html
pages/affiliate/index.html
pages/affiliate/login.html
pages/browse.html
pages/canceled.html
pages/contact.html
pages/customerservice.html
pages/flypage.html
pages/help.html
pages/index.html
pages/login.html
pages/logout.html
pages/modular_modify.html
pages/new_account.html
pages/ord/basket.html
pages/privacypolicy.html
pages/process_return.html
pages/quantity.html
pages/query/check_orders.html
pages/query/order_detail.html
pages/query/order_return.html
pages/results.html
pages/results_big.html
pages/returns.html
pages/saved_carts.html
pages/ship_addresses.html
pages/stock-alert-added.html
pages/stock-alert.html
pages/swap_results.html
```

30.8.8.1. Component Walkthrough -- category_vertical

The remainder of this section is best read while viewing the file CATROOT/templates/components/cart_display in a text editor.

Lines 1-6: Component Specification

```
 1  [comment]
 2  ui_component: category_vertical
 3  ui_component_group: Navigation
 4  ui_component_label: Vertical category list
 5
 6  page_class:
 7          label: Page class
 8          widget: select
 9          lookup: which_page
10          db: area
11          help: Defines which sets of items should be displayed
12          advanced: 1
```

```
13
14  set_selector:
15          label: Page area selector
16          widget: select
17          db: area
18          lookup: sel
19          help: Defines which sets of items should be displayed
20          advanced: 1
21  [/comment]
22
```

These lines control what is shown in the Edit page screen of the Administration Tool.

```
23  <tr><td align="center" class="categorybar">
24    <br>
25    <table>
26
27  <!-- BEGIN COMPONENT [control component category_vertical] -->
```

Lines 23−25 set up the row and table within that row that will hold the vertical category list. Line 27 identifies the start of the code for the list.

```
28  [loop
29          prefix=box
30          search="
31                  fi=area
32                  st=db
33                  tf=sort
34                  ac=0
35                  ac=0
36                  co=yes
37
38                  sf=sel
39                  op=eq
40                  se=[control set_selector left]
41
42                  sf=which_page
43                  op=rm
44                  se=[control page_class all|@@MV_PAGE@@]
45  "]
46
```

Lines 28−45 build a list of product categories obtained through a search of the area table.

```
47    <tr>
48      <td valign="top" class="categorybar">
49        <b>[box-exec bar_link]area[/box-exec]</b>
50      </td>
51    </tr>
52    <tr>
53      <td valign="top" class="categorybar">
54
55  [set found_cat][/set]
56  [loop prefix=cat
57          search="
58                  fi=cat
59                  st=db
60                  tf=sort
61                  tf=name
62                  rf=code,name
```

```
63                  sf=sel
64                  se=[box-code]
65          "
66          ]
67          [cat-exec bar_link]cat[/cat-exec]<BR>
68  [/loop]
69
70      </td>
71    </tr>
72  [/loop]
73
74    </table>
75    <br>
76  </td></tr>
77
78  <!-- END COMPONENT [control component category_vertical] -->
```

Lines 47−78 generate a list of links based on the products and product categories identified in the search.

30.8.9. cross_horizontal

The cross_horizontal component is used in the following pages:

```
pages/browse.html
pages/index.html
pages/results.html
pages/results_big.html
```

The cross_horizontal component is used in the following templates:

```
templates/foundation/cart
templates/foundation/leftonly
templates/foundation/leftright
```

30.8.10. cross_vertical

Not used in Foundation demo pages.

The cross_horizontal component is used in the following templates:

```
templates/foundation/leftright
```

30.8.11. modular_buy

The modular_buy component is used in the following pages:

```
pages/flypage.html
```

The modular_buy component is used in the following templates:

```
templates/components/modular_update
```

30.8.12. modular_update

The modular_update component is used in the following pages:

```
pages/modular_modify.html
```

30.8.13. promo

The promo component is used in the following pages:

```
pages/contact.html
pages/results_big.html
```

30.8.14. promo_horizontal

The promo_horizontal component is used in the following pages:

```
pages/aboutus.html
pages/canceled.html
```

The promo_horizontal component is used in the following templates:

```
templates/foundation/cart
templates/foundation/leftonly
templates/foundation/leftright
```

30.8.15. promo_vertical

Not used in Foundation demo pages.

The promo_horizontal component is used in the following templates:

```
templates/foundation/leftright
```

30.8.16. random

The random component is used in the following pages:

```
pages/browse.html
pages/index.html
pages/ord/basket.html
pages/privacypolicy.html
pages/process_return.html
pages/results.html
pages/swap_results.html
```

30.8.17. random_horizontal

Not used in Foundation demo pages.

The random_horizontal component is used in the following templates:

```
templates/foundation/cart
templates/foundation/leftonly
templates/foundation/leftright
```

30.8.18. random_vertical

Not used in Foundation demo pages.

The random_vertical component is used in the following templates:

```
templates/foundation/leftright
```

30.8.19. saved_carts_list_small

The saved_carts_list_small component is used in the following pages:

```
pages/ord/basket.html
```

30.8.20. search_box_small

The search_box_small component is used in the following pages:

```
pages/aboutus.html
pages/account.html
pages/affiliate/index.html
pages/affiliate/login.html
pages/browse.html
pages/canceled.html
pages/contact.html
pages/customerservice.html
pages/flypage.html
pages/help.html
pages/index.html
pages/login.html
pages/logout.html
pages/modular_modify.html
pages/new_account.html
pages/ord/basket.html
pages/privacypolicy.html
pages/process_return.html
pages/quantity.html
pages/query/check_orders.html
pages/query/order_detail.html
pages/query/order_return.html
pages/results.html
pages/results_big.html
pages/returns.html
pages/saved_carts.html
pages/ship_addresses.html
pages/stock-alert-added.html
pages/stock-alert.html
pages/swap_results.html
```

The search_box_small component is used in the following templates:

```
templates/regions/LEFTONLY_TOP
templates/regions/LEFTRIGHT_TOP
```

30.8.21. upsell

Not used in Foundation demo pages.

30.8.22. upsell_horizontal

The upsell_horizontal component is used in the following pages:

```
pages/flypage.html
```

The upsell_horizontal component is used in the following templates:

```
templates/foundation/cart
templates/foundation/leftonly
templates/foundation/leftright
```

30.8.23. upsell_vertical

Not used in Foundation demo pages.

The upsell_vertical component is used in the following templates:

```
templates/foundation/leftright
```

31. The Database Tables

Interchange catalogs are centralized around the database. You can alter any of the standard databases, add new databases, or remove unneeded databases

The foundation catalog includes the following tables, organized here by content:

- Your site content data
  area.txt
  cat.txt
  downloadable.txt
  merchandising.txt
  options.txt
  pricing.txt
  products.txt
  products.txt.category
- Customer data
  access.asc
  gift_certs.txt
  userdb.txt
- Transaction−related data
  inventory.txt
  orderline.txt
  order_returns.txt
  transactions.txt
- Third−party relationship data
  affiliate.txt
  banner.txt
- Site administrative data
  component.txt
  files.txt
  ichelp.txt
  icmenu.txt
  locale.txt
  mv_metadata
  route.txt
  shipping.asc
  variable.txt
- Ancillary data
  2ndDayAir.csv
  450.csv
  country.txt
  Ground.csv
  NextDayAir.csv
  salestax.asc
  state.txt

The following dictionary lists and describes each table used in the Foundation demo.

31.1. 2ndDayAir.csv

Shipping table from UPS (http://www.ups.com/using/services/rave/rate/). This and all shipping tables should be updated periodically.

31.2. 450.csv

Shipping table from UPS for 450xx Zip Code origin. You will probably need to get your own from the UPS site (http://www.ups.com/using/services/rave/rate/) and clip the headers.

31.3. Ground.csv

Shipping table from UPS (http://www.ups.com/using/services/rave/rate/).

31.4. NextDayAir.csv

Shipping table from UPS (http://www.ups.com/using/services/rave/rate/).

31.5. access.asc

Administrative access table. This table is used by the Administration Tool. For more description on these fields, see the Red Hat Interchange Administration Tool guide.

Fields

Field	*Description*
username	Login name or group name (group names begin with ':')
password	Hashed password
name	Administrator's name
last_login	Last login time
super	Set to 1 if superuser
yes_tables	Tables the user may edit
no_tables	Tables the user may not edit
upload	No Description
acl	No Description
export	No Description
edit	No Description
pages	No Description
files	No Description
config	No Description
reconfig	No Description
groups	Administrator's group memberships

meta	No Description
no_functions	Explicitly disallowed functions
yes_functions	Allowed functions with permission flags
table_control	No Description
personal_css	Administrator's personal CSS (for admin screen presentation)

31.5.1. username

Example Data

```
:ausers
:busers
BigUser
goody
ic
```

The login name for an administrator or an administration group. Group names are prefixed with a colon (':').

31.5.2. password

Example Data

```
Ksjs65bMNLjPQ
```

Hashed password.

31.5.3. name

Example Data

```
Interchange Site Administrator
Interchange Site Associates
Business Users
2nd Shift
Mr. Jones
Inbound Sales
```

Descriptive name for the administrator or administration group.

31.5.4. last_login

Example Data

```
989424489
```

Last login time (in unix time() format).

31.5.5. super

Boolean value. If true (1), the administrator has Interchange Site Administrator privilege.

31.5.6. yes_tables

Example Data

```
affiliate=vcx component=v gift_certs=v inventory=vx ...
NONE
```

Tables this administrator or administration group can access. This is a space−delimited list of 'table_name=permission_flags' entries.

31.5.7. no_tables

Example Data

```
access mv_metadata variable
```

Tables this administrator or administration group can not use. This is a space−delimited list of tables names.

31.5.8. upload

No Description

31.5.9. acl

No Description

31.5.10. export

No Description

31.5.11. edit

No Description

31.5.12. pages

No Description

31.5.13. files

No Description

31.5.14. config

No Description

Example Data

Allowed Values

31.5.15. reconfig

No Description

31.5.16. groups

Example Data

```
ausers
busers
```

Allowed Values

Groups the site user belongs to. You can set permissions for groups.

31.5.17. meta

No Description

31.5.18. no_functions

Example Data

```
orderstats trafficstats
```

Space−delimited list of functions explicitly not allowed for the site user.

31.5.19. yes_functions

Example Data

```
item=lvecd itemtype=lvc order=lvca orderstats trafficstats ...
NONE
```

Functions the site user can perform. This is a space−delimited list of functions, with permission flags if appropriate.

Usage examples

- dist/lib/UI/pages/admin/access_permissions.html

31.5.20. table_control

No Description

Usage examples

- dist/lib/UI/Primitive.pm
- dist/lib/UI/pages/admin/special/key_violation.html
- dist/lib/UI/usertag/if_mm

31.5.21. personal_css

Used in the Administration Tool screens to make personal changes to the page presentation. This is done by creating your own personal CSS (cascading style sheet).

Usage examples

- dist/lib/UI/pages/admin/preferences.html

31.6. affiliate

`cat_root/products/affiliate.txt`

This table contains data related to your affiliate programs. See also the affiliate_receptor component.

Fields

Field	*Description*
affiliate	Affiliate ID
name	Name of affiliate organization
campaigns	Campaigns this affiliate participates in
coupon_amount	Discount for customers from affiliate participating in coupon campaign
join_date	When the affiliate signed with you
url	Your default URL to use for customers coming from the affiliate site (not the affiliate's home page)
timeout	Timeout in seconds after which purchases are no longer credited to the affiliate
active	Boolean, set to 1 for active affiliates
password	Affiliate login password
image	Affiliate's logo

31.6.1. affiliate

Example Data

```
consolidated
hardhat
```

This field contains the unique Affiliate ID.

31.6.2. name

Example Data

```
Consolidated Diversified
Hardhat Construction
```

This is the descriptive name of the affiliate.

31.6.3. campaigns

Example Data

```
coupon
```

This field lists the campaigns that the affiliate participates and enables campaign features and tracks traffic from advertising campaigns. The foundation catalog implements a coupon campaign in the affiliate_receptor component. If you want to add campaigns, you will also need to develop the appropriate logic within the affiliate_receptor component and pages that use it.

31.6.4. coupon_amount

Example Data

```
5
```

This is the discount offered customers from the affiliate participating in the coupon campaign.

Note −− This is implemented in the affiliate_receptor component as a flat discount amount. If you wanted a percentage discount instead, you would modify the [discount] tag in catalog_root/templates/components/affiliate_receptor (see the [discount] tag for more detail).

31.6.5. join_date

Example Data

```
20000827
20000910
```

This is the date when the affiliate signed with you.

31.6.6. url

Example Data

```
http://demo.akopia.com/~hardhat
http://www.minivend.com/consolidated/
```

The value in this field is used to direct visitors coming from the Affiliate to your home page or a page you have designed specifically for visitors from that Affiliate's site. Note that this should *not* be the URL of the Affiliate's home site.

31.6.7. timeout

Example Data

```
0
3600
```

The value in this field is used to specify the amount of time a customer has to place an order to still give the Affiliate credit for it. If the customer goes over this amount of time, the Affiliate doesn't get credit for the customer visit. The timeout delay is measured in seconds, with the value of 0 (zero) disabling it. It is recommended that you use a value in the thousands to make sure the customer has enough time to shop.

31.6.8. active

This is a boolean value indicating whether the affiliate is active.

31.6.9. password

Example Data

```
akopia
```

Password for affiliate login (see catalog_root//pages/affiliate/login.html). Note that the password is stored in plaintext by default.

31.6.10. image

Example Data

```
http://demo.akopia.com/~hardhat/images/logo.gif
http://www.minivend.com/consolidated/conslogo.gif
```

Affiliate's logo image.

31.7. area

`cat_root/products/area.txt`

This table is used to implement dynamic navigation bars. For example, it is used in the category_horizontal and category_vertical components. Note the similarity to the **cat** table, since both **area** and **cat** tables supply data for building links to results pages.

When building entries in a navigation bar, it is the **bar_link** subroutine in the /dist/catalog_before.cfg configuration file that actually reads and processes the values from the table.

See also the following catalog and administrative templates:

- cat_root/templates/components/category_horizontal
- cat_root/templates/components/category_vertical
- dist/lib/UI/pages/admin/layout.html
- dist/lib/UI/pages/admin/layout_auto.html
- dist/lib/UI/pages/admin/wizard/do_launch.html
- dist/lib/UI/pages/admin/wizard/do_save.html

Fields

Field	***Description***

code	Unique key
sel	Space−delimited list of navigation bars to contain the entry
name	Display label
which_page	Page class in which the navigation bar may appear
sort	Sorting prefix for entry (preempts standard alphanumeric sort)
display_type	How to label links in the navbar (name, icon, url or image)
image	Image URL (if appropriate)
image_prop	HTML attributes for output <img> tag (if appropriate)
banner_image	Image name for use in target page
banner_text	Text for use in target page
link_type	Type of links in the navbar (external, internal, simple, complex)
url	Target for internal or external link_type
tab	Database table file to use with 'simple' **link_type**
page	Results page to use with 'simple' **link_type**
search	Search spec used with 'complex' **link_type**
selector	The selector used to scan the `products` table for products in the category
link_template	Overrides template used for building navbar links.

31.7.1. code

Example Data

```
1
2
3
```

Unique key.

31.7.2. sel

Example Data

```
left
```

Space−delimited list of navigation bars that should contain the entry. Note that comma or null should also work as a delimiter.

31.7.3. name

Example Data

```
Hand Tools
Hardware
Ladders
Measuring Tools
Painting Supplies
Safety Equipment
```

```
Specials
Tool Storage
```

Label to display.

31.7.4. which_page

Example Data

```
all
```

Page class in which the navigation bar may appear.

31.7.5. sort

Example Data

```
00
03
04
05
06
```

Lexographic (alphanumeric) sorting prefix. Note use of '03' rather than '3', which would sort after '13'. This controls the order of the categories in your navigation bar.

If this is not set, your navbar entries will sort in alphabetical order.

31.7.6. display_type

Example Data

```
name
icon
url
image
```

What to use for the labels in the navigation bar (for example, name, icon, url or image). The navigation bars in the foundation catalog are set up with 'name' display_type.

display type	***Link shown as***
name	Displays name only
icon	Displays name and specified image
image	Displays image only
url	Displays link

31.7.7. image

Image URL for image or icon **display_type**.

31.7.8. image_prop

For image or icon **display_type**, this contains the HTML attributes for the HTML that will appear in the navbar, for example:

```
<img src="image" alt="name" image_prop>name
```

31.7.9. banner_image

Example Data

```
promo_image.gif
```

This field is not related to banner ads. It is useful if you want to pass to your results page an image that is specific for the navbar entry (perhaps to display a banner above your results).

If you are using an Interchange search for your links (i.e., 'simple' or 'complex' **link_type**), then this will add 'va=banner_image=**banner_image**' to the resulting search specification. This puts the contents of **banner_image** into the Values hash of your search results page. You can access it with [value banner_image] (see the value tag). You will have to modify the standard results page (or set up and specify your own) in order to display the image.

```
<IMG src="[value banner_image]" alt="[value banner_text]">
```

The foundation catalog does not implement banner_image in the preconfigured navigation bars.

31.7.10. banner_text

Example Data

```
This Is A Title For Hand Tools
```

This field is not related to banner ads. It is useful if you want to pass to your results page some text that is specific for the navbar entry (perhaps to display a title above your results).

If you are using an Interchange search for your links (i.e., 'simple' or 'complex' **link_type**), then this will add 'va=banner_text=**banner_text**' to the resulting search specification. This puts the contents of **banner_text** into the Values hash of your search results page. You can access it with [value banner_text] (see the value tag). You will have to modify the standard results page (or set up and specify your own) in order to display the text.

The foundation catalog does not implement banner_text in the preconfigured navigation bars.

31.7.11. link_type

Example Data

```
none
external
internal
simple
```

```
complex
```

Link type to create in the navigation bar.

Link type	***Description***
none	No link
external	External link. The HTML specified in **url** will go directly into the navigation bar.
internal	Internal link. This will be highlit if it is the current page. If you specify both a page and a form for the link, the **url** field should contain "*page form*". See the Search Engine documentation for more detail on search forms.
simple	Allows you to specify an Interchange search with a few values. See the **bar_link** subroutine in the /dist/catalog_before.cfg configuration file for more detail.
complex	Allows you to fully specify an Interchange search specification. See the **bar_link** subroutine in the /dist/catalog_before.cfg configuration file for more detail if you need to use these.

31.7.12. url

Target URL (external link or internal page/search specification). See **link_type** . The foundation catalog navigation bars are not set up with link types that use the **url** field.

31.7.13. tab

Database table file to use with 'simple' **link_type** (searchspec fi=**tab**). The foundation catalog navigation bars are not set up with link types that use the **tab** field.

31.7.14. page

Results page to use with 'simple' **link_type** (searchspec sp=**page**). The foundation catalog navigation bars are not set up with link types that use the **page** field.

31.7.15. search

Search spec used with 'complex' **link_type**. See the Search Engine documentation for more detail on search forms. The foundation catalog navigation bars are not set up with link types that use the **search** field.

31.7.16. selector

The selector that is used to scan the `products` table for products in the category. Used with 'simple' **link_type**. The foundation catalog navigation bars are not set up with link types that use the **selector** field.

31.7.17. link_template

Overrides the usual HTML link template for navbar entries. See the **bar_link** subroutine in the /dist/catalog_before.cfg configuration file if you need to modify link templates.

The foundation catalog navigation bars are not set up with link types that use the **link_template** field.

31.8. banner

`cat_root/products/banner.txt`

The banner ad table. The foundation catalog does not implement any banner ads with this table.

You do not need to use this table to display ads served by third parties (for example, doubleclick). Since most banner ads on the internet are served by third parties and are not managed by your catalog, you probably will not need to set up banners here unless you do your own advertising.

See Banner/Ad rotation in the template documentation for a detailed description of the columns and content of the banner table. Also, see the `banner` tag documentation.

Fields

Field	*Description*
code	Key for the item. If the banners are not weighted, this should be a category−specific code.
category	Category for set of weighted banners
weight	Display frequency weight for weighted banner
rotate	Boolean: parse **banner** field for banners to rotate if true (1)
banner	Banner name or list of banners to rotate

31.8.1. code

Example Data

```
MyBanner
MyBanner2
MyBanner3
default
```

See Banner/Ad rotation.

31.8.2. category

Example Data

```
BannerCat1
```

See Banner/Ad rotation.

31.8.3. weight

Example Data

```
1
2
7
```

See Banner/Ad rotation.

31.8.4. rotate

Boolean value. If true (1), rotates banners listed in **banner**. See Banner/Ad rotation.

31.8.5. banner

Example Data

```
Default banner 1{or}Default banner 2{or}Default banner 3
First MyBanner
Second MyBanner
Third MyBanner
```

See Banner/Ad rotation.

31.9. cat

`cat_root/products/cat.txt`

This table contains properties of product categories. Notice the similarity to the **area** table, since both the **area** and **cat** tables supply data for building links to results pages.

Fields

Field	*Description*
code	Unique key
sel	Space−delimited list of foreign keys into **area** table
name	Category name
which_page	Page class in which the category may appear
sort	Sorting prefix for entry (preempts standard alphanumeric sort)
display_type	How to label the category links (name, icon, url or image)
image	Image URL (if appropriate)
image_prop	HTML attributes for output <img> tag (if appropriate)
banner_image	Image name for use in target page
banner_text	Text for use in target page
link_type	Type of links in the navbar (external, internal, simple, complex)
url	Target for internal or external link_type
tab	Database table file to use with 'simple' **link_type**
page	Results page to use with 'simple' **link_type**
search	Search spec used with 'complex' **link_type**
selector	The selector used to scan the `products` table for products in the category
link_template	Overrides template used for building links

31.9.1. code

Example Data

```
1
4
5
```

Unique key.

31.9.2. sel

Example Data

```
6
8 9
9
```

Space−delimited list of foreign key(s) into **area** table. The category will appear in each navbar section (defined by a row in the **area** table) where the key from **cat.sel** matches the **area.code**.

For example, the foundation catalog (tools) places Gift Certificates in more than one category of the left navbar.

31.9.3. name

Example Data

```
Breathing Protection
Eye Protection
Gift Certificate
Picks & Hatchets
Pliers
Rulers
Sandpaper
Toolboxes
```

Category name for display.

31.9.4. which_page

The page class. When building links, you can select categories matching a page class. This means you could set up your catalog to show a different list of links on page 'foo.html' than on page 'bar.html'.

31.9.5. sort

Example Data

```
01
03
```

Lexographic (alphanumeric) sorting prefix. Notice use of '03' rather than '3', which would sort after '13'. You

can use this to control the order of the categories in a list of links.

31.9.6. display_type

Example Data

```
name
icon
url
image
```

What to use for the labels in the navigation bar (for example, name, icon, url or image). The links in the foundation catalog are set up with 'name' display_type.

display type	***Link shown as***
name	Displays name only
icon	Displays name and specified image
image	Displays image only
url	Displays link

31.9.7. image

Image URL for image or icon **display_type**.

31.9.8. image_prop

For image or icon **display_type**, this contains the HTML <img ...> tag attributes for the links, for example:

```
<img src="image" alt="name" image_prop>name
```

31.9.9. banner_image

Example Data

```
promo_image.gif
```

This field is not related to banner ads. It is useful if you want to pass to your results page an image that is specific for the navbar entry (perhaps to display a banner above your results).

If you are using an Interchange search for your links (i.e., 'simple' or 'complex' **link_type**), then this will add 'va=banner_image=**banner_image**' to the resulting search specification. This puts the contents of **banner_image** into the Values hash in your search results page. You can access it with [value banner_image] (see the value tag). You will have to modify the standard results page (or set up and specify your own) in order to display the image. For example, you might include the following in your results page:

```
<IMG src="[value banner_image]" alt="[value banner_text]">
```

31.9.10. banner_text

This field is not related to banner ads. It is useful if you want to pass to your results page some text that is specific for the navbar entry (perhaps to display a title above your results).

If you are using an Interchange search for your links (i.e., 'simple' or 'complex' **link_type**), then this will add 'va=banner_text=**banner_text**' to the resulting search specification. This puts the contents of **banner_text** into the Values hash in your search results page. You can access it with [`value` banner_text] (see the `value` tag). You will have to modify the standard results page (or set up and specify your own) in order to display the text.

31.9.11. link_type

Example Data

```
none
external
internal
simple
complex
```

The link type to create.

Link type	***Description***
none	No link
external	External link. The HTML specified in **url** will go directly into the link.
internal	Internal link. This will be highlit if it is the current page. If you specify both a page and a form for the link, the **url** field should contain "*page form*". See the Search Engine documentation for more detail on search forms.
simple	Allows you to specify an Interchange search with a few values. See the **bar_link** subroutine in the /dist/catalog_before.cfg configuration file for more detail.
complex	Allows you to fully specify an Interchange search specification. See the **bar_link** subroutine in the /dist/catalog_before.cfg configuration file for more detail if you need to use these.

31.9.12. url

Target URL (external link or internal page/search specification). See **link_types** above.

31.9.13. tab

Example Data

```
products
```

Database table file to use with 'simple' **link_type** (searchspec fi=**tab**).

31.9.14. page

Example Data

```
swap_results
```

Results page to use with 'simple' **link_type** (searchspec sp=**page**).

31.9.15. search

Example Data

```
fi=merchandising^Msf=featured^Mse=new
fi=merchandising^Msf=featured^Mse=special^Msu=yes
```

Search spec used with 'complex' **link_type**. See the Search Engine documentation for more detail on search forms.

Note: The '^*M*' delimiters in the sample data represents a carriage return character (Control−M, or hexadecimal 0x0d).

31.9.16. selector

Example Data

```
category=Breathing Protection
category=Eye Protection
category=Gift Certificate
category=Picks & Hatchets
category=Pliers
category=Rulers
category=Sandpaper
category=Toolboxes
```

The element that is used to scan the `products` table for products in the category. Used with 'simple' **link_type**.

31.9.17. link_template

Overrides the usual HTML link template for navbar entries. See the **bar_link** subroutine in the /dist/catalog_before.cfg configuration file if you need to modify link templates.

31.10. country

A list of countries used to build select boxes and shipping mode choices based on countries.

```
code
sorder
region
selector
shipmodes
name
```

31.11. downloadable

This table controls downloadable products. The Marketing Reports data set for the foundation catalog demonstrates downloadable products. List a product's **sku** in this table if you want to deliver it through a download. A customer can then download the file specified in the **dl_location** field after checkout.

For reference, see the implementation in the following files:

- catalog_root/pages/deliver.html
- catalog_root/etc/receipt.html
- catalog_root/pages/query/order_detail.html

Fields

Field	*Description*
sku	Unique key, matches **product.sku**
dl_location	Location of downloadable file
dl_type	MIME type of downloadable file

31.11.1. sku

Example Data

Example Data from the Marketing Reports data set:

```
00352as
22083da
49503cg
59330rt
59402fw
73358ee
83491vp
90773sh
```

This is the unique key for this table that is also the common key into the products table.

31.11.2. dl_location

Example Data from 'reports' catalog

```
download/00352as.pdf
download/22083da.pdf
download/49503cg.pdf
download/59330rt.pdf
download/59402fw.pdf
download/73358ee.pdf
download/83491vp.pdf
download/90773sh.pdf
```

File location of downloadable product.

31.11.3. dl_type

Example Data from 'reports' catalog

```
application/pdf
```

MIME type of downloadable content.

31.12. files.txt

A database where files (pages, etc.) can be kept instead of in the Unix filesystem.

31.13. gift_certs.txt

```
code
username
order_date
original_amount
redeemed_amount
available_amount
passcode
active
redeemed
update_date
```

31.14. inventory.txt

```
sku
    Quantity info
quantity
            Gets decremented after each sale.
stock_message
            The usual shipping time of the product.
    Out of stock message:
        In stock
        Ships in 3-5 days
        Ships in 4-6 weeks
        Special order
account
    Accounting info
    Sales account
cogs_account
```

31.15. locale.txt

```
code
en_US
de_DE
fr_FR
```

31.16. merchandising.txt

```
sku
featured
```

```
banner_text
banner_image
blurb_begin
blurb_end
    Closer (end text for feature display)
timed_promotion
start_date
    Start date
finish_date
upsell_to
    Cross-sell SKUs
cross_sell
cross_category
others_bought
times_ordered
```

31.17. mv_metadata

See the following sections in the icadvanced catalog for more information:

- display tag and mv_metadata
- mv_metadata.asc

31.18. options

This table contains data for implementing simple, matrix and modular options.

Simple options are options that a customer can combine arbitrarily, such as size and color. The selected options might affect price. See the `accessories` tag for more detail on option values for simple options.

Matrix options are preconfigured combinations of options. For example, if you sell titanium and carbon−fiber bike frames, but offer only certain combinations of frame material and color, your checkout page might include a select box with only the following entries:

- Silver Titanium: $1672
- Black Titanium: $1672
- Red Titanium: $1674
- Black Carbon Fiber: $1290
- Yellow Flame Carbon Fiber: $1300

Note that there is no Yellow Flame Titanium offering, for example.

Modular options are like a structured bill of materials, where one product is a master item and other products are subitems for that master item. The subitems can also be master items to subitems at a lower level. In addition, subitems may be designated as 'phantom', which means that they are placeholders in the hierarchy of the structured bill of materials with their own subitems, but are not actual items themselves.

The foundation catalog with the computer data set uses modular options.

For more information, see the following pages and components in the foundation catalog:

- cat_root/pages/flypage.html

- cat_root/templates/components/modular_buy
- cat_root/templates/components/modular_update

Note: Subsequent foundation catalog releases may place simple, matrix, and modular option types in separate tables.

Fields

Field	***Description***
code	Unique ID for the product option
o_master	SKU of the master item for the option
sku	SKU for the option (foreign key into products table)
o_group	Product grouping code
o_sort	Sorting prefix for list display
phantom	Boolean −− Item is a phantom placeholder (as in structured bill of materials) with suboptions.
o_enable	Boolean −− enables suboptions for the option
o_matrix	Matrix−type option (preconfigured combinations of attributes)
o_modular	Modular−type option (master/subitem relationship like modular bill of materials)
o_default	Default selection for the option group or suboption for a phantom option
o_label	Short name for option display
o_value	Simple option values (in Interchange option format)
o_widget	The HTML widget to use for displaying the option group
o_footer	Not used in foundation catalog
o_header	Not used in foundation catalog
o_height	Height of widget (if applicable)
o_width	Width of widget
description	Option/Variant description (for description in display)
price	Price of this option/variant
wholesale	Dealer price of this option/variant
differential	Differential to add to the base item price when using a phantom bill of materials
weight	Weight difference with this option/variant (for shipping)
volume	Volume difference with this option/variant
mv_shipmode	No Description
o_exclude	Option groups to exclude (trumped by **o_include**). Modular only.
o_include	Option groups to include (trumps **o_exclude**). Modular only.

31.18.1. code

Example Data

```
1002
1003
```

```
1004
1005
```

Unique ID for the option.

31.18.2. o_master

Example Data

```
00010
999000
999001
999002
```

SKU of the master item for the option. The master item is one level up in the modular hierarchy, and must be one of the following:

- An item in the products table (matching **products.sku**)
- Another option in the options table (matching **options.sku**)
- A phantom item in the **options** table.

If an option has a master item, then a customer can not choose that option without having previously selected the master item.

The price for a master item is the sum of the master item's price and the price for each of the subitems. Because the subitems are recursively defined, the top−level item reflects the top level price plus the price of all selected options.

31.18.3. sku

Example Data

```
00010
999000
7000015
7000030
```

The sku for the item or option. This may not be unique for matrix options or if an option that belongs to multiple **o_master**s is listed for each master.

31.18.4. o_group

Example Data

```
A
B
C
I
```

Product group (scanned to see whether it applies to this product or not)

31.18.5. o_sort

Example Data

```
01
02
03
04
47
48
49
50
```

Sorting prefix for listing order of options.

31.18.6. phantom

Modular options only.

Boolean −− if true (1), then this is a phantom item acting as a placeholder for other items rather than an actual product. The item's **sku** will not match an entry in the products table, though the **o_master** will match either the **sku** of another phantom item in the **options** table or the **sku** of an item in the **products** table.

31.18.7. o_enable

Boolean −− Enables subitems for this item or option. Note that an option with **o_enable** false may itself still be a subitem for an option or item above it.

31.18.8. o_matrix

Boolean. Set true (1) for matrix−type options. See the options table in the tools data set for examples of matrix options. Matrix options that are part of a set have the same value for **options.sku**.

31.18.9. o_modular

Specifies a modular option. See main heading for description of modular options.

31.18.10. o_default

Example Data

```
1
11002
7000062
7000087
```

Selects the default option for a group.

31.18.11. o_label

Example Data

```
Add a second hard drive
Case Color
Case color
Case style
Include tapes
Red
```

This is the short name for option display.

31.18.12. o_value

Example Data

```
1=One 8GB tape,\r2=Two 8GB tapes,\r=None*
a=One 8GB tape,\rb=Two 8GB tapes,\r=None*
baby=Baby Tower,\rmid=Mid-tower,\rfull=Full Tower
baby=Baby tower,\rmid=Mid-tower,\rfull=Full tower
red=Passion Red,\rblue=Electric Blue,\rgreen=Sea Green,\rgrey=S...
red=Passion Red,\ryellow=Lemon Yellow,\rblue=Electric Blue,\rgr...
red=Rage Red,\ryellow=Honey Yellow
```

This is an Interchange value set for a simple option. It is typically a comma−delimited list of labels and values with '*' indicating the default value. See the `accessories` tag for more detail.

Note that the "\r" characters in the above example represent carriage returns in the actual data ("\r" in perl, or Ctrl−M, or hexadecimal 0D), and the ... indicates a line too long to show.

31.18.13. o_widget

Example Data

```
select
```

This determines the HTML Widget type (*e.g.*, a select box). For example, the [`options`] tag uses this entry when building HTML widgets in a page. See also the [`accessories`] tag for available widgets.

31.18.14. o_footer

Example Data

Allowed Values

31.18.15. o_header

Example Data

Allowed Values

31.18.16. o_height

This allows you to set the height of the HTML widget, if appropriate.

31.18.17. o_width

This allows you to set the width of the HTML widget, if appropriate.

31.18.18. description

Example Data

```
ATX Mid Tower−Grey (3)5.25 (2)3.5 & (1)3.5 Hidden
Enlight ATX Desktop Case (2)5.25 & (2)3.5
Enlight ATX Tower Case (4) 5.25 & (2)3.5
Micro ATX Tower − Honey Yellow
Micro ATX Tower − Moody Blue
Micro ATX Tower − Rage Red
Micro ATX Tower − Smoky Grey
Super Tower Case (6)5.25 & (3)3.5
```

Longer description to show when displaying the options.

31.18.19. price

Example Data

```
0.00
10
20
29
75
```

This sets the retail price of the option.

31.18.20. wholesale

Example Data

```
13
40.00
```

This sets the dealer price of the option.

31.18.21. differential

Example Data

```
−209
−40
−79
```

The phantom bill of materials for an option group can have a differential, which is an amount to add to the base price of the master product to get to a new base price that accommodates the phantom bill of materials. Note that the differential can be negative.

For example, in the computer data set of the foundation catalog, SKU 00011 in the products table is an $849.95 pre−configured Athlon 800MHz computer that includes a 17" monitor (in this case, SKU 7000087 in the products table).

The monitor by itself would otherwise have cost $209. It is much more convenient if you can use the same option part number and price for each item. To do this, you need a phantom option (in this case, SKU 999105 in the options table only) with a differential of −209 and the available monitors as suboptions. When you include the phantom option in the bill of materials for the computer (SKU 00011), the $−209 differential adjustment makes the price work out properly.

For instance, suppose that a $499 computer is configured as follows:

```
500 MHz Athlon          --  $499
32 MB SDRAM             --  ZERO
10 GB disk              --  ZERO

TOTAL                   --  $499
```

Suppose it costs $90 to upgrade the base computer to 128M of RAM and $150 for a 30 GB hard disk.

If you also sell an 128MB 800 MHz $899 computer, and the customer upgrades to the 30 GB hard disk,

```
800 MHz Athlon           --  $899
(memory differential)    --  $-90
128 MB RAM               --   $90
30 GB  disk              --  $150

TOTAL                    -- $1039
```

If you did not have the differential, you would need a different option part number for each item make the number come out right.

With the differential, you can use the same part number for 128MB RAM no matter what the base part is. The price is always $90 −− there is just a −90 differential when ordered with the 800MHz Athlon, making the effective price zero.

31.18.22. weight

Example Data

```
5
```

Shipping weight of the option. Interchange uses this to calculate shipping cost.

31.18.23. volume

Volume added by the option.

31.18.24. mv_shipmode

No Description

31.18.25. o_exclude

Modular options only.

Lists the option groups to exclude once the include has been done. Takes the form of a number of wildcard atoms.

31.18.26. o_include

Modular options only.

Lists the option groups to include with your item. Takes the form of a number of wildcard atoms.

31.19. order_returns.txt

```
code
order_number
session
username
rma_number
nitems
total
return_date
update_date
```

31.20. orderline.txt

Every line item that is actually ordered is detailed in this table. The order as a whole is one record in the transactions table.

See the page `query/check_orders.html` for how it can be used. See `etc/report` for how to add to it.

```
code
store_id
order_number
session
username
shipmode
sku
quantity
price
subtotal
shipping
taxable
mv_mi
mv_si
size
color
options
```

```
order_date
update_date
status
        pending = Pending
        shipped = Shipped
        backorder = Back ordered
        credit = Waiting for credit check
        canceled = Cancelled
parent
affiliate
campaign
description
mv_mp
```

31.21. pricing

This database works in conjunction with the `CommonAdjust` directive to allow quantity pricing for one product or for a group of products (sometimes known as mix−and−match). The fields `q2, q5, q10, etc.` are for the quantity levels; the `price_group` field selects the mix−and−match category for the product.

Fields

Field	*Description*
sku	Unique key, shared with products table
price_group	Mix−and−match category
q2	Retail, 2 or more
q5	Retail, 5 or more
q10	Retail, 10 or more
q25	Retail, 25 or more
q100	Retail, 100 or more
w2	Wholesale, 2 or more
w5	Wholesale, 5 or more
w10	Wholesale, 10 or more
w25	Wholesale, 25 or more
w100	Wholesale, 100 or more

31.21.1. sku

Example Data

```
os28004
os28006
os28057c
os28069
```

Unique key, matching the **sku** for an entry in products table.

31.21.2. price_group

Example Data

```
general
```

This field determines mix−and−match categories if you want to allow mix−and−match quantity pricing (i.e., where 5 of *these* plus 5 of *those* afford the **q10** price for both *these* and *those*).

31.21.3. q2

If set, this will be the price per item when the order quantity is 2 or greater.

31.21.4. q5

If set, this will be the price per item when the order quantity is 5 or greater.

31.21.5. q10

If set, this will be the price per item when the order quantity is 10 or greater.

31.21.6. q25

If set, this will be the price per item when the order quantity is 25 or greater.

31.21.7. q100

If set, this will be the price per item when the order quantity is 100 or greater.

31.21.8. w2

If set, this will be the dealer price per item when the order quantity is 2 or greater.

31.21.9. w5

If set, this will be the dealer price per item when the order quantity is 5 or greater.

31.21.10. w10

If set, this will be the dealer price per item when the order quantity is 10 or greater.

31.21.11. w25

If set, this will be the dealer price per item when the order quantity is 25 or greater.

31.21.12. w100

If set, this will be the dealer price per item when the order quantity is 100 or greater.

31.22. products

This is the main table for product data. See also 'The Product Database' section in the database documentation.

The **sku** is also the master key in many of the related tables.

Fields

Field	*Description*
sku	Unique product ID
description	Short description for list display
title	Full title for book, CD, artwork, *etc.*
template_page	Not used in foundation catalog. No description.
comment	Longer description for item display (e.g., flypage.html)
thumb	Thumbnail image
image	Regular−sized image
price	Retail quantity one price
wholesale	Dealer minimum quantity price
prod_group	Product supercategory
category	Product category
nontaxable	Boolean. Set true (1) if nontaxable
weight	Weight in your units. Should match shipping table.
size	List of options used with accessories tag.
color	List of options used with accessories tag.
gift_cert	Boolean. Set true (1) if this is a gift certificate.
related	Deprecated in favor of **merchandising.upsell_to**
featured	Deprecated. Use **merchandising** table.
inactive	Boolean. Set true (1) to inactivate a product
url	Not Documented

31.22.1. sku

Example Data

```
gift_cert
os28004
os28006
os28057c
```

Unique identifier for the product. You should use only characters of the class `A−Z a−z 0−9 _ −` (i.e., matching the regular expression, '[−A−Za−z0−9_]+'). Although Interchange itself does not impose this

restriction, you may have problems with SQL databases, file systems, and URL encoding if you use other characters. For example, a slash (/) can interfere with URLs and filenames, a colon (:) can interfere with database representations (or file names on some operating systems), i<etc.>

31.22.2. description

Example Data

```
Brush Set
Disposable Brush Set
Ergo Roller
Gift Certificate
Painters Brush Set
Painters Ladder
Spackling Knife
Trim Brush
```

A short description for the product that is used for displaying in the shopping cart, receipt, and order report.

31.22.3. title

Example Data

```
Brush Set
Disposable Brush Set
Ergo Roller
Gift Certificate
Painters Brush Set
Painters Ladder
Spackling Knife
Trim Brush
```

This column is not used in the foundation catalog. Previously used in the Art store (simple) demo for a painting title. You probably want to use **description** instead.

You should modify the products and other tables to suit your catalog's requirements. You might use this field if you want to show titles for books, music, or other titled merchandise. If you do not use a title that is distinct from the short description, then you probably do not need this column in the table at all.

31.22.4. template_page

Not used in foundation catalog.

No Description.

31.22.5. comment

Example Data

```
A must have for all painters!  This spackling knife is ergon...
Enjoy the perfect feel and swing of our line of hammers. Thi...
This set includes 2" and 3" trim brushes and our ergonomical...
This set of disposable foam brushes is ideal for any stainin...
```

This is the field for a long description of the product. If you are using an Interchange text/gdbm database, the field size is unlimited; if using another type of database, the length will be dependent on the field type selected. If you are using a SQL database, see the appropriate cat_root/dbconf subdirectory for a place to set COLUMN_DEF values. See also the database documentation, '[Importing from an ASCII File]', for details about defaults for columns that you do not define.

31.22.6. thumb

Example Data

```
gift_certificate.gif
os28004_b.gif
os28005_b.gif
os28006_b.gif
```

This is the filename for a small (thumbnail) image of the product.

31.22.7. image

Example Data

```
gift_certificate_large.gif
os28004.gif
os28005.gif
os28006.gif
```

This is the filename for a regular−sized image of the product, as it should appear in an HTML <img src="**image**"> tag. You do not need to specify the path if the image files are in the usual Interchange image directory.

31.22.8. price

Example Data

```
1.00
12.99
14.99
9.99
```

The quantity−one price of the product. See the **wholesale** field and the price table for dealer and quantity pricing.

31.22.9. wholesale

Example Data

```
1
10
11
12
```

This is the minimum dealer price for the item. For quantity pricing, see the price table.

31.22.10. prod_group

Example Data

```
Hand Tools
Hardware
Ladders
Measuring Tools
Miscellaneous
Painting Supplies
Safety Equipment
Tool Storage
```

Product group (supercategory). This indicates the grouping of product categories, for example in the navigation bars created from the **area** table (note the match with the **name** data in the area table).

31.22.11. category

Example Data

```
Brushes
Gift Certificate
Hammers
Ladders
Nails
Paintbrushes
Putty Knives
Rollers
```

This is the category the product should appear in when you select a list. You can put a product in more than one category, but you may need to accommodate this in display and banner headings. Embedded perl is helpful for this.

31.22.12. nontaxable

Boolean value. If true (1), the sales tax calculation for an order will not include the cost of the product. See also the `salestax` tag.

31.22.13. weight

Example Data

```
1
2
3
```

This is a numeric value of the weight used for determining shipping costs (with UPS, for example). In the US, this is typically the weight in pounds in order to match the UPS, Fed Ex and other standard shipping tables.

31.22.14. size

Example Data

```
1", 2", 3"
1', 1.5'
1/4", 1/2", 3/4", 1", 2", 3"
10oz, 15oz, 20 oz
2"
6'
set
standard, metric
```

This is where the old Construct Something demo store kept the 'size' options for a product. The foundation catalog now uses the **options** table instead to handle product options (also sometimes called product attributes).

The `accessories` tag can build HTML widgets from the comma−delimited list of product options. You can use a delimiter other than comma (if compatible with the table) as long as you also set the `delimiter` in the `accessories` tag.

You probably do not need this field if you use the options table (for example, if you are building from the foundation catalog).

31.22.15. color

Another product option column. No longer used in the foundation catalog. See **size** above for description.

31.22.16. gift_cert

Boolean value. If true (1), specifies that this product is a gift certificate. See also the **gift_certs** table.

31.22.17. related

Used for displaying "upsells," opportunities to purchase an additional item when this one is purchased. Contains a comma−separated list of SKUs to be offered.

The foundation catalog now instead uses the **upsell_to** field of the **merchandising** table for upselling.

31.22.18. featured

Deprecated in favor of the **merchandising** table.

31.22.19. inactive

If true (1), renders the product inactive (i.e., it will not appear in the catalog).

31.22.20. url

Not Documented

31.23. products.category.txt

The products.category.txt table (actually a link to products.txt.10) is an automatically generated index into the products table to speed category searches. See Dictionary Indexing With INDEX in the database documentation for details about auto−indexing of text databases.

31.24. route.txt

```
code
attach
continue
commit
commit_tables
counter
credit_card
cyber_mode
email
empty
encrypt
encrypt_program
errors_to
increment
inline_profile
individual_track
individual_track_ext
partial
pgp_cc_key
pgp_key
profile
receipt
reply
report
rollback
rollback_tables
supplant
track
```

31.25. salestax.asc

31.26. shipping.asc

Shipping methods table

31.27. state.txt

State/territory/county information

```
code
sorder
country
state
name
tax
postcode
shipmodes
tax_name
```

31.28. transactions.txt

Each individual customer order has an entry in this table. The line items are not entered here, but in the orderline table.

See the page `query/check_orders.html` for how it can be used. See `etc/report` for how to add to it.

```
code
store_id
order_number
session
username
shipmode
nitems
subtotal
shipping
handling
salestax
total_cost
fname
lname
    Last Name
company
address1
address2
    Address line 2
city
state
zip
country
phone_day
    Daytime Phone
phone_night
    Home Phone
fax
email
b_fname
b_lname
    Billing Last Name
b_company
b_address1
b_address2
    Billing Address Line 2
b_city
b_state
    Billing State
b_zip
    Billing Postcode
b_country
    Billing Country
b_phone
order_date
order_ymd
order_wday
payment_method
po_number
avs
order_id
update_date
```

```
status
affiliate
campaign
parent
archived
deleted
complete
comments
```

31.29. userdb.txt

The user database used for maintaining customer address information, account information, preferences, and more. See icdatabase for more information.

```
username
password
acl
mod_time
s_nickname
company
fname
lname
address1
address2
address3
city
state
zip
    Postcode
country
    Country
phone_day
mv_shipmode
b_nickname
b_fname
b_lname
b_address1
b_address2
b_address3
b_city
b_state
b_zip
b_country
b_phone
    Billing Phone
mv_credit_card_type
mv_credit_card_exp_month
mv_credit_card_exp_year
p_nickname
email
fax
phone_night
fax_order
    Payment method:
        (none) = Credit Card
        1 = Fax or Mail
        2 = Purchase order
        3 = COD
address_book
accounts
```

```
preferences
carts
owner
file_acl
db_acl
order_numbers
email_copy
mail_list
    Mailing lists the customer has joined:
        offer = Special offers
        newsletter = Newsletter
        alert = Alerts and Recalls
        upgrade = Upgrades
project_id
account_id
order_dest
credit_limit
inactive
dealer
    Dealer:
        (none) = No
        1 = Yes
b_company
feedback
    ???
```

31.30. variable.txt

Configuration database

```
code
    Variable name
Variable
pref_group
    Preferences area
```

32. HTML Hypertext links

Normally, regular hypertext links are not used in Interchange pages. These kinds of links will not include the session ID. If the customer follows an external link back to the catalog, the list of products ordered so far will have been lost. The `area` tag is used to generate a hypertext link which includes a session ID.

Instead of:

```
<A HREF="/cgi-bin/mv/shirts">Shirts</A>
```

Use:

```
<A HREF="[area shirts]">Shirts</A>
```

33. Images

Inline images are placed in Interchange pages in the normal fashion with <IMG SRC="URL">. But since Interchange pages are served by a CGI program, do not use relative links. The Foundation store defines an image directory with the `ImageDir` and `ImageDirSecure` directives that automatically adjusts the image path to a set base directory.

34. Browser Cookies

The Foundation store enables the `Cookies` directive so that users with cookie−capable browsers will retain session context. Then, standard HREF and Interchange page links can be intermixed without the fear of losing the shopping basket. Cookie capability is also required to use search caching, page caching, and statically generated pages. If the user's browser does not support cookies, the cache will be ignored.

If planning to use more than one host name within the same domain for naming purposes (perhaps a secure server and non−secure server), set the domain with the `CookieDomain` directive. This must contain at least two periods (.) as per the cookie specification, and must be located in the same server as the domain.

35. Dependencies in administration

In general, it's a good idea to leave fields empty if you don't want to use them, instead of removing them from the database altogether. That way nothing in admin or demo will break. line:

Template Guide

36. 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.

36.1. 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.

37. 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 envoke 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 *Red Hat Interchange 4.8: Development Guide*.

38. Using Interchange Template Tags

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

38.1. 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 follwoing 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 ...], [sql ...], and more. 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]"]
```

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

```
[loop list="code        field    field2  field3
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]`). `[page ...]` and `[order ..]` are **not** container tags.

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 before being passed the container text. Exceptions include `[calc]` .. `[/calc]` and `[currency]` ... `[/currency]`. All tags accept the INTERPOLATE=1 tag modifier, which causes the interpretation to take place. It is not necessary to interpret the contents of a `[set variable]` TAGS `[/set]` pair, as they might contain tags which should only be upon evaluating an order profile, search profile, or `mv_click` operation. If the evaluation is performed at the time a variable is set, use `[set name=variable interpolate=1] TAGS [/set]`.

38.2. The DATA and FIELD Tags

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.

38.3. set, seti, 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 three tags that are used to set the space, `[set name]value[/set]`, [seti name]value[/seti], [tmp name]value[/tmp], and two variations (or shortcuts).

[set variable]value[/set]

named attributes: `[set name="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 something][/seti]

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

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

[tmp name]value[/tmp]

The same as [seti] but it does not persist.

[scratch name]

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

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

[scratchd]

The same as [scratch name], except it deletes the value. 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 for more information.

38.4. 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`. See `catalog.cfg`. Both can be sorted with `[sort table:field:mod −start +number]` modifiers. See `Sorting`.

[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-field]

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

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

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

Outputs the IF if the `field` in the `products` table is not empty, and 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 ...] 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−data 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, or −1), 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, or −1), 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.

38.5. if

[if type field op* compare*]

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 textwill 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, and combine conditions:

```
[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.

39. 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.

39.1. 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.

39.2. 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 arg=values]

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

[/perl]
```

39.3. 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.

39.4. 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).

40. Interchange Perl Objects

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');
```

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

```
<% $Document->write("Value of foo is $CGI->{foo}"); %>
```

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 `[mvasp tables="foo"]` parameter to the tag call. Once the database is shared, it is open and can be accessed by any of its methods. This will not work with SQL unless AllowGlobal is set for the catalog.

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

```
$ref = $Db{products};
```

The available methods are:

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

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

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

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

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

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

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

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

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

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

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

# Create a subroutine to set a slice of the database
$sub = $ref->row_settor(@columns);
for (@keys) {
    $sub->($key, @values);
}

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

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

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

%Sql

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

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

This will not work with unless AllowGlobal is set for your catalog.

$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 that has several routines associated with it.

```
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 ASP call. For example:

HTML style:

```
<HTML MV="mvasp" MV.TABLES="products pricing">
```

or
Named parameters:

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

or
Positional parameters:

```
[mvasp products pricing]
```

Any tag can be called.

```
<%
    my $user = $Session->{username};
    my $name_from_db = $Tag->data('userdb', 'name', $user );
    $Document->write($name_from_db);
%>
```

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 supress 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");
%>
```

41. 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/mvdebug
```

41.1. 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
```

41.2. 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);
```

41.3. 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]
code: [value mv_order_number]
shipping_mode: [shipping-description]
status: pending
[/import]

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

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

41.4. 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
```

41.5. Header

41.6. 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.

41.7. 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.

41.8. 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
```

41.9. 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.

41.10. 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.

[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.

[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.

42. 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.

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

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
```

42.2. 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/backet.htm)

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.

42.3. 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 minivend.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.

43. 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.

43.1. 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_cache_params          S   Determines caching of searches
mv_change_frame          A   Any form, changes frame target of form output
mv_check                 A   Any form, sets multiple user variables after update
mv_checkout              O   Sets the checkout page
mv_click                 A   Any form, sets multiple form variables before update
mv_click                 XA  Default mv_click routine, click is mv_click_arg
mv_click <name>          XA  Routine for a click <name>, sends click as arg
mv_click_arg             XA  Argument name in scratch space
mv_coordinate        co  S   Enables field/spec matching coordination
mv_column_op         op  S   Operation for coordinated search
mv_credit_card*          O   Discussed in order security (some are read-only)
mv_delay_page        dp  S   Delay search until after inital page display
```

mv_dict_end	de	S	Upper bound for binary search
mv_dict_fold	df	S	Non-case sensitive binary search
mv_dict_limit	di	S	Sets upper bound based on character position
mv_dict_look	dl	S	Search specification for binary search
mv_dict_order	do	S	Sets dictionary order mode
mv_doit		A	Sets default action
mv_email		O	Reply-to address for orders
mv_exact_match	em	S	Sets word-matching mode
mv_failpage	fp	O,S	Sets page to display on failed order check/search
mv_field_file	ff	S	Sets file to find field names for Glimpse
mv_field_names	fn	S	Sets field names for search, starting at 1
mv_first_match	fm	S	Start displaying search at specified match
mv_head_skip	hs	S	Sets skipping of header line(s) in index
mv_index_delim	id	S	Delimiter for search fields (TAB default)
mv_matchlimit	ml	S	Sets match page size
mv_max_matches	mm	S	Sets maximum match return (only for Glimpse)
mv_min_string	ms	S	Sets minimum search spec size
mv_negate	ne	S	Records NOT matching will be found
mv_nextpage	np	A	Sets next page user will go to
mv_numeric	nu	S	Comparison numeric in coordinated search
mv_order_group		O	Allows grouping of master item/sub item
mv_order_item		O	Causes the order of an item
mv_order_number		O	Order number of the last order (read-only)
mv_order_quantity		O	Sets the quantity of an ordered item
mv_order_profile		O	Selects the order check profile
mv_order_receipt		O	Sets the receipt displayed
mv_order_report		O	Sets the order report sent
mv_order_subject		O	Sets the subject line of order email
mv_orsearch	os	S	Selects AND/OR of search words
mv_profile	mp	S	Selects search profile
mv_range_alpha	rg	S	Sets alphanumeric range searching
mv_range_look	rl	S	Sets the field to do a range check on
mv_range_max	rx	S	Upper bound of range check
mv_range_min	rm	S	Lower bound of range check
mv_record_delim	dr	S	Search index record delimiter
mv_return_all	ra	S	Return all lines found (subject to range search)
mv_return_delim	rd	S	Return record delimiter
mv_return_fields	rf	S	Fields to return on a search
mv_return_file_name	rn	S	Set return of file name for searches
mv_return_spec	rs	S	Return the search string as the only result
mv_save_session		C	Set to non-zero to prevent expiration of user session
mv_search_field	sf	S	Sets the fields to be searched
mv_search_file	fi	S	Sets the file(s) to be searched
mv_search_line_return	lr	S	Each line is a return code (loop search)
mv_search_match_count		S	Returns the number of matches found (read-only)
mv_search_page	sp	S	Sets the page for search display
mv_searchspec	se	S	Search specification
mv_searchtype	st	S	Sets search type (text, glimpse, db or sql)
mv_separate_items		O	Sets separate order lines (one per item ordered)
mv_session_id	id	A	Suggests user session id (overridden by cookie)
mv_shipmode		O	Sets shipping mode for custom shipping
mv_sort_field	tf	S	Field(s) to sort on
mv_sort_option	to	S	Options for sort
mv_spelling_errors	er	S	Number of spelling errors for Glimpse
mv_substring_match	su	S	Turns off word-matching mode
mv_successpage		O	Page to display on successful order check
mv_todo		A	Common to all forms, sets form action
mv_todo.map		A	Contains form imagemap
mv_todo.checkout.x		O	Causes checkout action on click of image
mv_todo.return.x		O	Causes return action on click of image
mv_todo.submit.x		O	Causes submit action on click of image
mv_todo.x		A	Set by form imagemap

```
mv_todo.y                A   Set by form imagemap
mv_unique             un  S   Return unique search results only
mv_value              va  S   Sets value on one-click search (va=var=value)
```

43.2. Form Actions

Interchange form processing is based on an `action` and a `todo`. The predefined actions at the first level are:

```
process       process a todo
search        form-based search
scan          path-based search
order         order an item
minimate      get access to a database via MiniMate
```

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.

43.3. 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]
```

43.4. 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>
```

43.5. 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.

43.6. 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.

43.7. 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−order]">
<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−order]">
[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.

43.8. 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.

43.9. 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 `"`, 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.

43.9.1. 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.

44. Internationalization

Interchange has a rich set of internationalization (I18N) features that allow conditional message display, differing price formats, different currency definitions, price factoring, sorting, and other settings. The definitions are maintained in the catalog.cfg file through the use of built−in POSIX support and Interchange's `Locale` directive. All settings are independent for each catalog and each user visiting that catalog, since customers can access the same catalog in an unlimited number of languages and currencies.

44.1. Setting the Locale

The locale could be set to `fr_FR` (French for France) in one of two ways:

[setlocale locale=locale* currency=locale* persist=1*]

This tag is for new−style tags only and will not work for `[old]`.
Immediately sets the locale to `locale`, and will cause it to persist in future user pages if the `persist` is set to a non−zero, non−blank value. If the `currency` attribute is set, the pricing and currency−specific locale keys and Interchange configuration directives are modified to that locale. If there are no arguments, it sets it back to the user's default locale as defined in the scratch variables `mv_locale` and `mv_currency`.
This allows:

```
Dollar Pricing:

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

Franc Pricing:

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

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

[page process/locale/fr_FR/page/catalog]

This is the same as `[page catalog]`, except when the link is followed it will set the locale to `fr_FR` before displaying the page. This is persistent.

[page process/locale/fr_FR/currency/en_US/page/catalog]

This is the same as `[page catalog]`, except when the link is followed it will set the locale to `fr_FR` and the pricing/number display to the locale `en_US` before displaying the page. This is persistent.

Once the locale is persistently set for a user, it is in effect for the duration of their session.

44.2. Interchange Locale Settings

The `Locale` directive has many possible settings that allow complete internationalization of page sets and currencies. The `Locale` directive is defined in a series of key/value pairs with a key that contains word characters only being followed by a value. The value must be enclosed in double quotes if it contains whitespace. In this example, the key is `Value setting`.

```
Locale fr_FR "Value setting" "Configuration de valeur"
Locale de_DE "Value setting" Werteinstellung
```

When accessed using the special tag `[L]Value setting[/L]`, the value `Configuration de valeur` will be displayed **only** if the locale is set to `fr_FR`. If the locale is set to `de_DE`, the string `Werteinstellung` will be displayed. If it is neither, the default value of `Value setting` will be displayed.

The `[L]` and `[/L]` must be capitalized. This is done for speed of processing as well as easy differentiation in text.

Another, way to do this is right in the page. The `[LC]` ... `[/LC]` pragma pair permits specification of locale−dependent text.

```
[LC]
          This is the default text.
  [fr_FR] Text for the fr_FR locale. [/fr_FR]
  [de_DE] Text for the de_DE locale. [/de_DE]
[/LC]
```

You can also place an entirely new page in place of the default one if the locale key is defined. When a locale is in force, and a key named `HTMLsuffix` is set to that locale, Interchange first looks for a page with a suffix corresponding to the locale. For example:

<A HREF="[area index]">Catalog home page</A>

If a page index.html exists, it will be the default. If the current locale is `fr_FR`, a page "index.fr_FR" exists, and `Locale` looks like this:

```
Locale fr_FR HTMLsuffix  fr_FR
```

Then, the `.fr_FR` page will be used instead of the `.html` page. For a longer series of strings, the configuration file recognizes:

```
Locale fr_FR <<EOF
{
    "Value setting",
    "Configuration de valeur",

    "Search",
    "Recherche"
}
EOF
```

This example sets two string substitutions. As long as this is a valid Perl syntax describing a series of settings, the text will be matched. It can contain any arbitrary set of characters that don't contain `[L]` and `[/L]`. If

using double quotes, string literals like \n and \t are recognized.

A database can also be used to set locale information. Locale information can be added to any database in the `catalog.cfg` file, and the values in it will overwrite previous settings. For more information, see LocaleDatabase. The `[L]default text[/L]` is set before any other page processing takes place. It is equivalent to the characters "default text" or the appropriate `Locale` translation for all intents and purposes. Interchange tags and Variable values can be embedded.

Because the `[L] message [/L]` substitution is done before any tag processing, the command `[L][item-data table field][/L]` will fail. There is an additional `[loc] message [/loc]` *UserTag* supplied with the distribution. It does the same thing as `[L] [/L]` except it is programmed after all tag substitution is done. See the interchange.cfg.dist file for the definition.

Note: Be careful when editing pages containing localization information. Even changing one character of the message can change the key value and invalidate the message for other languages. To prevent this, use:

```
[L key]The default.[/L]
```

The key `msg_key` will then be used to index the message. This may be preferable for many applications.

A `localize` script is included with Interchange. It will parse files included on the command line and produce output that can be easily edited to produce localized information. Given an existing file, it will merge new information where appropriate.

44.3. Special Locale Keys for Price Representation

Interchange honors the standard POSIX keys:

```
mon_decimal_point    or      decimal_point
mon_thousands_sep    or      thousands_sep
currency_symbol      or      int_currency_symbol
frac_digits  or      p_cs_precedes
```

See the POSIX setlocale(3) man page for more information. These keys will be used for formatting prices and approximates the number format used in most countries. To set a custom price format, use these special keys:

price_picture

Interchange will format a currency number based on a "picture" given to it. The basic form is:

```
Locale en_US price_picture "$ ###,###,###.##"
```

The `en_US` locale, for the United States, would display `4452.3` as `$ 4,452.30`. The same display can be achieved with:

```
Locale en_US mon_thousands_sep ,
Locale en_US mon_decimal_point .
Locale en_US p_cs_precedes     1
Locale en_US currency_symbol   $
```

A common price_picture for European countries would be `###.###.###,##`, which would display that same number as `4.452,30`. To add a franc notation at the end for the locale `fr_FR`, use the setting:

```
Locale fr_FR price_picture "##.###,## fr"
```

IMPORTANT NOTE: The decimal point in use, set by `mon_decimal_point`, and the thousands separator, set by `mon_thousands_sep` must match the settings in the price_picture. The `frac_digits` setting is not used in this case. It is derived from the location of the decimal (if any).

The same setting for `fr_FR` above can be achieved with:

```
Locale fr_FR mon_thousands_sep .
Locale fr_FR mon_decimal_point ,
Locale fr_FR p_cs_precedes     0
Locale fr_FR currency_symbol   fr
```

If the number of digits is greater than the # locations in the price_picture, the digits will be changed to asterisks. An overflow number above would show as `**.***,** fr`.

picture

Same as `price_picture`, but sets the value returned if the `[currency]` tag is not used. If the number of digits is greater than the # locations in the picture, the digits will be changed to asterisks, displaying something like `**,***.**`.

44.4. Dynamic Locale Directive Changes

If a Locale key is set to correspond to an Interchange `catalog.cfg` directive, that value will be set when the locale is set.

PageDir

To use a different page directory for different locales, set the `PageDir` key. For example, to have two separate language page sets, French and English, set:

```
# Establish the default at startup
PageDir   english
Locale fr_FR  PageDir  francais
Locale en_US  PageDir  english
```

ImageDir

To use a different image directory for different locales, set the `ImageDir` key. To have two separate language button sets, French and English, set:

```
# Establish the default at startup
ImageDir   /images/english/
Locale fr_FR  ImageDir   /images/francais/
Locale en_US  ImageDir   /images/english/
```

ImageDirSecure

See ImageDir.

PriceField

To use a different field in the products database for pricing based on locale, set the `PriceField` locale setting. For example:

```
# Establish the default at startup
PriceField    price
Locale fr_FR  PriceField  prix
```

The default will always be `price`, but if the locale `fr_FR` is set, the `PriceField` directive will change to `prix` to give prices in francs instead of dollars.
If `PriceBreaks` is enabled, the `prix` field from the `pricing` database will be used to develop the quantity pricing.

Note: If no `Locale` settings are present, the display will always be `price`, regardless of what was set in `PriceField`. Otherwise, it will match `PriceField`.

PriceDivide

Normally used to enable penny pricing with a setting of 100, `PriceField` can be used to do an automatic conversion calculation factor based on locale.

```
# Default at startup is 1 if not set
# Franc is strong these days!
Locale fr_FR  PriceDivide  .20
```

The price will now be divided by `.20`, making the franc price five times higher than the dollar.

PriceCommas

This controls whether the `mon_thousands_sep` will be used for standard currency formatting. This setting will be ignored if you are using `price_picture`. Set the value to 1 or 0, to enable or disable it. Do not use yes or no.

```
# Default at startup is Yes if not set
PriceCommas  Yes
Locale fr_FR  PriceCommas  0
Locale en_US  PriceCommas  1
```

UseModifier

Changes the fields from the set shopping cart options.

```
# Default at startup is 1 if not set
# Franc is strong these days!
UseModifier format
Locale fr_FR  UseModifier formats
```

If a previous setting was made for an item based on another locale, it will be maintained.

PriceAdjustment

Changes the fields set by `UseModifier` that will be used to adjust pricing for an automatic conversion factor based on locale. For example:

```
# Default at startup
PriceAdjustment  format
Locale fr_FR  PriceAdjustment  formats
```

TaxShipping,SalesTax

Same as the standard directives.

DescriptionField

This changes the field accessed by default with the `[item-description]` and `[description code]` tags. For example

```
# Establish the default at startup
DescriptionField    description
Locale fr_FR  DescriptionField desc_fr
```

The [locale] tag

Standard error messages can be set based on Locale settings. Make sure not to use any of the predefined keys. It is safest to begin a key with msg_ . The default message is set between the `[locale key]` and `[/locale]` tags. See the example above.

44.5. Sorting Based on Locale

The Interchange `[sort database:field]` keys will use the LC_COLLATE setting for a locale provided that:

- The operating system and C compiler support locales for POSIX, and have the locale definitions set.
- The locale setting matches any configured locales.

If this arbitrary database named `letters`:

```
code        letter
00-0011     f
99-102      é
19-202      a
```

and this loop:

```
[loop 19-202 00-0011 99-102]
[sort letters:letter]
[loop-data letters letter]   [loop-code]
[/loop]
```

used the default C setting for LC_COLLATE, the following would be displayed:

```
a  19-202
f  00-0011
é  99-102
```

If the proper LC_COLLATE settings for locale `fr_FR` were in effect, then the above would become:

```
a  19-202
é  99-102
f  00-0011
```

44.6. Placing Locale Information in a Database

Interchange has the capability to read its locale information from a database, named with the `LocaleDatabase` directive. The database can be of any valid Interchange type. The locales are in columns, and the keys are in rows. For example, to set up price information:

```
key                 en_US   fr_FR   de_DE
PriceDivide         1       .1590   .58
mon_decimal_point   .       ,       ,
mon_thousands_sep   ,       .
currency_symbol     $        frs    DM
ps_cs_precedes      1       0       0
```

This would translate to the following:

```
Locale en_US PriceDivide         1
Locale en_US mon_decimal_point   .
Locale en_US mon_thousands_sep   ,
Locale en_US currency_symbol     $
Locale en_US ps_cs_precedes      1

Locale fr_FR PriceDivide         .1590
Locale fr_FR mon_decimal_point   ,
Locale fr_FR mon_thousands_sep   .
Locale fr_FR currency_symbol     " frs"
Locale fr_FR ps_cs_precedes      0

Locale de_DE PriceDivide         .58
Locale de_DE mon_decimal_point   ,
Locale de_DE mon_thousands_sep   " "
Locale de_DE currency_symbol     "DM "
Locale de_DE ps_cs_precedes      1
```

These settings append and overwrite any that are set in the catalog configuration files, including any include files.

Important note: This information is only read during catalog configuration. It is not reasonable to access a database for translation or currency conversion in the normal course of events. line:

Interchange Databases

45. Databases and Interchange

Interchange can use GDBM, DB_File, SQL, LDAP, or in−memory databases. In most cases, these different database formats should operate the same when called by Interchange's access methods.

Also, Interchange does not require an external SQL database. If you have a small database and do not want to integrate your own tool set, you might want to use Interchange's internal database. However, the order management functions of Interchange will be slower and not as robust without an SQL database. SQL is strongly recommended for at least the orderline, transactions, and userdb tables.

Keeping a database in an SQL manager makes it easier to integrate Interchange with other tools. Interchange can be used to maintain a spreadsheet containing product information through modifying the file `products.txt` as needed. References to SQL, DBI, and DBD can be ignored.

45.1. Text Source Files

Interchange reads delimited text files to obtain data. However, the text files are not the database. They are the source information for the database tables.

By default, all database source files are located in the `products` subdirectory of the catalog directory. The main products database is in the `products/products.txt` file in the supplied demo catalog.

Note: If you are using one of the internal database methods, any changes made to the ASCII source file will be reflected in the database in the next user session. If the product database contains less than a thousand records, updates will be instantaneous. If the product database is larger, updates will take longer. Use the `NoImport` reference tag to stop auto updating.

In the following configuration directive:

```
Database  products  products.txt   TAB
```

the `products` table will obtain its source information from the file `products.txt`. What is done with it depends on the type of underlying database being used. The different types and their behavior are described below:

GDBM

The database source file is checked to see if it is newer than the actual database file, `products.gdbm`. If it is, the database table is re−imported from the file.
This behavior can be changed in a few ways. If files should not be imported unless the `.gdbm` file disappears, set the `NoImport` directive:

```
NoImport  products
```

If the database source file is only to be imported at catalog start−up time, use the IMPORT_ONCE modifier:

```
Database products IMPORT_ONCE 1
```

GDBM is the default database type if the GDBM_File Perl module is installed (as it is on LINUX).

DB_File

The database source file is checked to see if it is newer than the actual database file, `products.db`. If it is, the database table is re−imported from the file. You can change this behavior in the same way as GDBM_File, described above.
DB_File is the default database type if the GDBM_File Perl module is not installed. This is common on FreeBSD. To specify DB_File as your database type, set it in `catalog.cfg` with a Database directive:

```
Database  products  DB_FILE   1
```

DBI/SQL

> If a file named `products.sql` is in the same directory as `products.txt`, the database table will not be imported from the ASCII source. If there is no `products.sql`, the following will occur: DBI/SQL imports will only happen at catalog configuration time.

Interchange will connect to the SQL database using the specified DSN. (DBI parameter meaning "Database Source Name".)
The table will be dropped with "DROP TABLE products;". This will occur without warning. NOTE: This can be prevented in several ways. See `NoImport External` or the SQL documentation for more information.
The table will be created. If there are COLUMN_DEF specifications in `catalog.cfg`, they will be used. Otherwise, the key (first field in the text file by default) will be created with a `char(16)` type and all other fields will be created as `char(128)`. The table creation statement will be written to the `error.log` file.
The text source file will be imported into the SQL database. Interchange will place the data in the columns. Data typing must be user−configured. This means that if "none" is placed in a field, and it is defined as a numeric type, the database import will not succeed. And if it does not succeed, the catalog will not become active.

In−Memory

Every time the catalog is configured, the `products.txt` file is imported into memory and forms the database. Otherwise, the database is not changed. The in−memory database is the default database if there is no GDBM_File or DB_File Perl module installed; specify it with:

```
Database   products   MEMORY   1
```

45.2. Interchange Database Conventions

This section describes naming and file usage conventions used with Interchange.

Note: Throughout the documentation, the following terms and their definitions are used interchangeably:

---

key, code

A reference to the database key. In Interchange, this is usually the product code or SKU, which is the part number for the product. Other key values may be used to generate relationships to other database tables.
It is recommended that the key be the first column of the ASCII source file, since Interchange's import, export, and search facilities rely on this practice.

field, column

The vertical row of a database. One of the columns is always the key and it is usually the first one.

table, database

A table in the database. Because Interchange has evolved from a single−table database to an access method for an unlimited number of tables (and databases, for that matter), a table will occasionally be referred to as a database. The only time the term database refers to something different is when describing the concept as it relates to SQL, where a database contains a series of tables. While Interchange cannot create SQL databases, it can drop and create tables with that database if given the proper permissions.

If necessary, Interchange can read the data to be placed in tables from a standard ASCII−delimited file. All of the ASCII source files are kept in the products directory, which is normally in the catalog directory (where catalog.cfg is located). The ASCII files can have ^M (carriage return) characters, but must have a new line character at the end of the line to work. NOTE: Mac users uploading files must use ASCII mode, not binary mode.

Interchange's default ASCII delimiter is TAB.

Note: The items must be separated by a single delimiter. The items in this document are lined up for reading convenience.

TAB

Fields are separated by `^I` characters. No whitespace is allowable at the beginning of the line.

```
code    description             price   image
SH543   Men's fine cotton shirt 14.95   shirts.jpg
```

PIPE

Fields are separated by pipe `|` characters. No whitespace is allowable at the beginning of the line.

```
code|description|price|image
SH543|Men's fine cotton shirt|14.95|shirts.jpg
```

CSV

Fields are enclosed in quotes, separated by commas. No whitespace should be at the beginning of the line.

```
"code","description","price","image"
"SH543","Men's fine cotton shirt","14.95","shirts.jpg"
```

Note: Using the default TAB delimiter is recommended if you plan on searching the ASCII source file of the database. PIPE works fairly well, but CSV delimiter schemes might cause problems with searching.

IMPORTANT NOTE: Field names are usually case−sensitive. Use consistency when naming or you might encounter problems. All lower or all upper case names are recommended.

Interchange uses one mandatory database, which is referred to as the products database. In the supplied demo catalog, it is called products and the ASCII source is kept in the file `products.txt` in the products directory. This is also the default file for searching with the THE SEARCH ENGINE. Interchange also has a two of standard, but optional, databases that are in fixed formats:
shipping.asc

The database of shipping options that is accessed if the `CustomShipping` directive is in use. This is a fixed−format database, and must be created as specified. For more information, see the Shipping ITL tag in the *Red Hat Interchange 4.8: Tag Reference Guide*.
salestax.asc

The database of sales tax information if the `[salestax]` tag is to be used. A default is supplied. NOTE: Caution, these things change! This is a fixed−format database, and must be created as specified. See Sales Tax.
These are never stored in SQL or DBM.

45.3. The Product Database

Each product being sold should be given a product code, usually referred to as SKU, a short code that identifies the product on the ordering page and in the catalog. The products.txt file is a ASCII−delimited list of all the product codes, along with an arbitrary number of fields which must contain at least the fields `description` and `price` (or however the `PriceField` and `DescriptionField` directives have been set). Any additional information needed in the catalog can be placed in any arbitrary field. See Interchange Database Capability for details on the format.

Field names can be case−sensitive depending on the underlying database type. Unless there are fields with the names "description" and "price" field, set the `PriceField` and `DescriptionField` directives to use the `[item−price]` and `[item−description]` tags.

The product code, or SKU, must be the first field in the line, and must be unique. Product codes can contain the characters **A−Za−z0−9**, along with hyphen (−), underscore (_), pound sign/hash mark (#), slash (/), and period (.). Note that slash (/) will interfere with on−the−fly page references. Avoid if at all possible.

The words should be separated by one of the approved delimiting schemes (TAB, PIPE, or CSV), and are case−sensitive in some cases. If the case of the "description" or "price" fields have been modified, the `PriceField` and `DescriptionField` directives must be appropriately set.

Note: CSV is not recommended as the scheme for the products database. It is much slower than TAB− or PIPE−delimited, and dramatically reduces search engine functionality. No field−specific searches are possible. Using CSV for any small database that will not be searched is fine.

IMPORTANT NOTE: The field names must be on the first line of the `products.txt` file. These field names must match exactly the field names of the `[item−field]` tags in the catalog pages, or the Interchange server will not access them properly. Field names can contain the characters A−Za−z0−9 and underscore (_).

More than one database may be used as a products database. If the catalog directive, ProductFiles, is set to a space−separated list of valid Interchange database identifiers, those databases will be searched (in the order specified) for any items that are ordered, or for product information (as in the `[price code]` and `[field`

code] tags).

When the database table source file (i.e., products.txt) changes after import or edit, a DBM database is re−built upon the next user access. No restart of the server is necessary.

If changing the database on−the−fly, it is recommended that the file be locked while it is being modified. Interchange's supplied import routines do this.

45.4. Multiple Database Tables

Interchange can manage an unlimited number of arbitrary database tables. They use the TAB delimiter by default, but several flexible delimiter schemes are available. These are defined by default:

```
Type 1      DEFAULT − uses default TAB delimiter
Type 2      LINE
            Each field on its own line, a blank line
            separates the record. Watch those carriage
            returns! Also has a special format when CONTINUE
            is set to be NOTES.
Type 3      %%
            Fields separated by a \n%%\n combination, records by
            \n%%%\n (where \n is a newline). Watch those carriage
            returns!
Type 4      CSV
Type 5      PIPE
Type 6      TAB
Type 7      reserved
Type 8      SQL
```

The databases are specified in `Database` directives, as:

```
Database    arbitrary arbitrary.csv CSV
```

This specifies a Type 4 database, the ASCII version of which is located in the file `arbitrary.csv`, and the identifier it will be accessed under in Interchange is "arbitrary." The DBM file, if any, will be created in the same directory if the ASCII file is newer, or if the DBM file does not exist. The files will be created as `arbitrary.db` or `arbitrary.gdbm`, depending on DBM type.

The `identifier` is case sensitive, and can only contain characters in the class [A−Za−z0−9_]. Fields are accessed with the [item_data `identifier` field] or [data `identifier` field key] elements. NOTE: Use of lower−case letters is strongly recommended.

If one of the first six types is specified, the database will automatically be built in the default Interchange DB style. The type can be specified with DB_FILE, GDBM, or MEMORY, if the type varies from that default. They will coexist with an unlimited number of DBI databases of different types.

In addition to the database, the session files will be kept in the default format, and are affected by the following actions.

The order of preference is:

GDBM

This uses the Perl `GDBM_File` module to build a GDBM database. The following command will indicate if GDBM is in Perl:

```
perl −e 'require GDBM_File and print "I have GDBM.\n"'
```

Installing GDBM_File requires rebuilding Perl after obtaining the GNU GDBM package, and is beyond the scope of this document. LINUX will typically have this by default; most other operating systems will need to specifically build in this capability.

DB_File (Berkeley DB)

This uses the `DB_File` module to build a Berkeley DB (hash) database. The following command will indicate if DB_File is in Perl:

```
perl −e 'require DB_File and print "I have Berkeley DB.\n"'
```

Installing `DB_File` requires rebuilding Perl after obtaining the Berkeley DB package, and is beyond the scope of this document. BSDI, FreeBSD, and LINUX will typically have it by default; most other operating systems will need to specifically build this in.
If using `DB_File`, even though `GDBM_File` is in Perl, set the environment variable MINIVEND_DBFILE to a true (non−zero, non−blank) value:

```
# csh or tcsh
setenv MINIVEND_DBFILE 1

# sh, bash, or ksh
MINIVEND_DBFILE=1 ; export MINIVEND_DBFILE
```

Then, re−start the server.
Or, to set a particular table to use Berkeley DB, the DB_FILE class in `catalog.cfg` can be specified:

```
Database arbitrary  DB_FILE  1
```

In−memory

This uses Perl hashes to store the data directly in memory. Every time the Interchange server is restarted, it will re−import all in−memory databases for every catalog.
If this is used, despite the presence of `GDBM_File` or `DB_File`, set the environment variable MINIVEND_NODBM as above or specify the memory type in the Database directive:

```
Database arbitrary  MEMORY  1
```

Note: The use of memory databases is not recommendeded.

45.5. Character Usage Restrictions

To review, database identifiers, field names, and product codes (database keys) are restricted in the characters they may use. The following table shows the restrictions:

	Legal characters
Database identifiers	A−Z a−z 0−9 _

```
Field names                         A-Z a-z 0-9 _
Database keys (product code/SKU)    A-Z a-z 0-9 _ # - . /
Database values                     Any (subject to field/record delimiter)
```

Some SQL databases have reserved words which cannot be used as field names; Interchange databases do not have this restriction.

For easy HTML compatibility, it is not recommended that a / be used in a part number if using the flypage capability. It can still be called [page href=flypage arg="S/KU"].

45.6. Database Attributes

Especially in SQL databases, there are certain functions that can be set with additional database attributes. For text import, the CONTINUE extended database import attribute allows additional control over the format of imported text.

Note: CONTINUE applies to all types except CSV. (Do not use NOTES unless using type LINE.)

CONTINUE

One of UNIX, DITTO, LINE, NONE, or NOTES. The default, NONE, is to simply split the line/record according to the delimiter, with no possible spanning of records. Setting CONTINUE to UNIX appends the next line to the current when it encounters a backslash (\) at the end of a record, just like many UNIX commands and shells.
DITTO is invoked when the key field is blank. It adds the contents of following fields to the one above, separated by a new line character. This allows additional text to be added to a field beyond the 255 characters available with most spreadsheets and flat-file databases.
Example in catalog.cfg:

```
Database products products.txt  TAB
Database products CONTINUE      DITTO
```

Products.asc file:

```
code     price     description
00-0011  500000    The Mona Lisa, one of the worlds great masterpieces.
                   Now at a reduced price!
```

The description for product 00-0011 will contain the contents of the `description` field on both lines, separated by a new line.

Note: Fields are separated by tabs, formatted for reading convenience.

This will work for multiple fields in the same record. If the field contains any non-empty value, it will be appended.
LINE is a special setting so a multi-line field can be used. Normally, when using the LINE type, there is only data on one line separated by one blank line. When using CONTINUE LINE, there may be some number of fields which are each on a line, while the last one spans multiple lines up until the first blank line.
Example in catalog.cfg:

```
Database products products.txt  LINE
Database products CONTINUE      LINE
```

Products.asc file:

```
code
price
description

00-0011
500000
The Mona Lisa, one of the worlds great masterpieces.
Now at a reduced price!

00-0011a
1000
A special frame for the Mona Lisa.
```

NOTES reads a Lotus Notes "structured text" file. The format is any number of fields, all except one of which must have a field name followed by a colon and then the data. There is optional whitespace after the colon. Records are separated by a settable delimiting character which goes on a line by itself, much like a "here document." By default, it is a form feed (^L) character. The final field begins at the first blank line and continues to the end of the record. This final field is named `notes_field`, unless set as mentioned below. Interchange reads the field names from the first paragraph of the file. The key field should be first, followed by other fields in any order. If one (and only one) field name has whitespace, then its name is used for the `notes_field`. Any characters after a space or TAB are used as the record delimiter.
If there are none, then the delimiter returns to the default form feed (^L) and the field name reverts to `notes_field`. The field in question will be discarded, but a second field with whitespace will cause an import error. Following records are then read by name, and only fields with data in them need be set. Only the `notes_field` may contain a new line. It is always the last field in the record, and begins at the **first** blank line.
The following example sets the delimiter to a tilde `(~)` and renames the `notes_field` to `description`. Example in catalog.cfg:

```
Database products products.txt  LINE
Database products CONTINUE      NOTES
```

Products.asc file:

```
code
title
price
image
description ~
size
color

title: Mona Lisa
price: 500000
code: 00-0011
image: 00-0011.jpg

The Mona Lisa, one of the worlds great masterpieces.
Now at a reduced price!
~
title: The Art Store T-Shirt
code: 99-102
```

```
size: Medium, Large*, XL=Extra Large
color: Green, Blue, Red, White*, Black
price: 2000

Extra large 1.00 extra.
~
```

EXCEL

Microsoft Excel is a widely−used tool to maintain Interchange databases, but has several problems with its standard TAB−delimited export, like encasing fields containing commas in quotes, generating extra carriage returns embedded in records, and not including trailing blank fields. To avoid problems, use a text−qualifier of none.
Set the EXCEL attribute to 1 to fix these problems on import:

```
Database products EXCEL 1
```

This is normally used only with TAB−delimited files.

45.7. Dictionary Indexing With INDEX

Interchange will automatically build index files for a fast binary search of an individual field. This type of search is useful for looking up the author of a book based on the beginning of their last name, a book title based on its beginning, or other similar situations.

Such a search requires a dictionary ordered index with the field to be searched contained in the first field and the database key (product code) in the second field. If the `INDEX field` modifier is specified, Interchange will build the index upon database import:

```
Database  products  products.txt   TAB
Database  products  INDEX          title
```

If the `title` field is the fourth column in the `products` database table, a file `products.txt.4` will be built, containing two tab−separated fields something like:

```
American Gothic   19−202
Mona Lisa         00−0011
Sunflowers        00−342
The Starry Night  00−343
```

Options can be appended to the field name after a colon (:). The most useful will be `f`, which does a case−insensitive sort. The `mv_dict_fold` option must be added to the search in this case.

Another option is `c`, which stands for "comma index." To index on comma−separated sub−fields within a field, use the :c option:

```
Database  products  products.txt   TAB
Database  products  INDEX          category:c
```

This can get slow for larger databases and fields. Interchange will split the field on a comma (stripping surrounding whitespace) and make index entries for each one. This allows multiple categories in one field while retaining the fast category search mechanism. It might also be useful for a `keywords` field.

The fast binary search is described in greater detail in THE SEARCH ENGINE below.

45.8. MEMORY for Memory−Only Databases

Interchange's memory−based databases are the fastest possible way to organize and store frequently used data. To force a database to be built in memory instead of DBM, use the MEMORY modifier:

```
Database  country  country.asc   TAB
Database  country  MEMORY        1
```

Obviously, large tables will use a great deal of memory, and the data will need to be re−imported from the ASCII source file at every catalog reconfiguration or Interchange restart. The big advantage of using MEMORY is that the database remains open at all times and does not need to be reinitialized at every connect. Use it for smaller tables that will be frequently accessed.

The MEMORY modifier forces IMPORT_ONCE.

45.9. IMPORT_ONCE

The IMPORT_ONCE modifier tells Interchange not to re−import the database from the ASCII file every time it changes. Normally, Interchange does a comparison of the database file modification time with the ASCII source every time it is accessed, and if the ASCII source is newer it will re−import the file.

IMPORT_ONCE tells it only to import on a server restart or catalog reconfiguration:

```
Database  products  products.txt   TAB
Database  products  IMPORT_ONCE    1
```

SQL databases don't normally need this. They will only be imported once in normal operation. Also see `NoImport` for a way to guarantee that the table will never be imported.

IMPORT_ONCE is always in effect for MEMORY databases. A catalog reconfiguration is required to force a change.

45.10. Importing in a Page

To add a data record to a database as a result of an order or other operation, use Interchange's `[import ...]` tag.

[import table type*] RECORD [/import]

Named parameters:

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

Import one or more records into a database. The `type` is any of the valid Interchange delimiter types, with the default being TAB. The table must already be a defined Interchange database table. It cannot be created

on−the−fly. If on−the−fly functionality is need, it is time to use SQL.

The import type selected need not match the type the database was specified. Different delimiters may be used.

The `type` of `LINE` and `continue` setting of `NOTES` is particularly useful, for it allows fields to be named and not have to be in any particular order of appearance in the database. The following two imports are identical in effect:

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

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

The `code` or key must always be present, and is always named `code`. If `NOTES` mode is not used, the fields must be imported in the same order as they appear in the ASCII source file.

The `file` option overrides the container text and imports directly from a named file based in the catalog directory. To import from `products.txt`, specify `file="products/products.txt"`. If the NoAbsolute directive is set to `Yes` in minivend.cfg, only relative path names will be allowed.

The `[import ....] TEXT [/import]` region may contain multiple records. If using `NOTES` mode, a separator must be used, which, by default, is a form−feed character (^L). See Import Attributes for more information.

45.11. Exporting from a Database

To export an existing database to a file suitable for searching by Interchange, create a page that contains a `[tag export ...][/tag]` element. Perhaps a better method is to define the same sort of tags in an OrderProfile, and use forms and buttons to access the profile.

45.12. Write Control

Interchange databases can be written in the normal course of events, either using the `[import ...]` tag or with a tag like `[data table=table column=field key=code value=new−value]`. To control writing of a global database, or to a certain catalog within a series of subcatalogs, or make one read only, see the following:

To enable write control:

```
Database   products  WRITE_CONTROL  1
```

Once this is done, to make a database read only, which won't allow writing even if `[tag flag write]products[/tag]` is specified:

```
Database   products  READ_ONLY  1
```

To have control with `[tag flag write]products[/tag]`:

```
Database   products  WRITE_TAGGED  1
```

To limit write to certain catalogs, set:

```
Database   products  WRITE_CATALOG  simple=0, sample=1
```

The "simple" catalog will not be able to write, while "sample" will if `[tag flag write]products[/tag]` is enabled. If a database is to always be writable, without having to specify `[tag flag write] ... [/tag]`, then define:

```
Database   products  WRITE_ALWAYS  1
```

The default behavior of SQL datbases is equivalent to WRITE_ALWAYS, while the default for GDBM_File, DB_File, and Memory databases is equivalent to:

```
Database   products  WRITE_CONTROL 1
Database   products  WRITE_TAGGED  1
```

45.13. Global Databases

If a database is to be available to all catalogs on the Interchange server, it may be defined in `minivend.cfg`. Any catalog running under that server will be able to use it. It is writable by any catalog unless WRITE_CONTROL is used.

46. SQL Support

Interchange can use any of a number of SQL databases through the powerful Perl DBI/DBD access methods. This allows transparent access to any database engine that is supported by a DBD module. The current list includes mSQL, MySQL, Solid, PostgreSQL, Oracle, Sybase, Informix, Ingres, Dbase, DB2, Fulcrum, and others. Any ODBC (with appropriate driver) should also be supported.

No SQL database is included with Interchange, but there are a number widely available on the Internet. Most commonly used with Interchange are PostgreSQL, MySQL, and Oracle. It is beyond the scope of this document to describe SQL or DBI/DBD. Sufficient familiarity is assumed.

In most cases, Interchange cannot perform administrative functions, like creating a database or setting access permissions. This must be done with the tools provided with a SQL distribution. But, if given a blank database and the permission to read and write it, Interchange can import ASCII files and bootstrap from there.

46.1. SQL Support via DBI

The configuration of the DBI database is accomplished by setting attributes in additional Database directives after the initial defining line as described above. For example, the following defines the database **arbitrary** as a DBI database, sets the data source (DSN) to an appropriate value for an mSQL database named `minivend` on port 1114 of the local machine:

```
Database arbitrary arbitrary.asc SQL
Database arbitrary DSN           dbi:mSQL:minivend:localhost:1114
```

As a shorthand method, include the DSN as the type:

```
Database arbitrary arbitrary.asc dbi:mSQL:minivend:localhost:1114
```

Supported configuration attributes include (but are not limited to):

DSN

A specification of the DBI driver and its data source. To use the DBD::mSQL driver for DBI, use:

```
dbi:mSQL:minivend:othermachine.my.com:1112
```

where mSQL selects the driver (case IS important), `minivend` selects the database, `othermachine.my.com` selects the host, and 1112 is the port. On many systems, `dbi:mSQL:minivend` will work fine. Of course, the `minivend` database must already exist.
This is the same as the DBI_DSN environment variable, if the DSN parameter is not set. Then, the value of DBI_DSN will be used to try and find the proper database to connect to.

USER

The user name used to log into the database. It is the same as the environment variable **DBI_USER**. If a user name is not needed, just don't set the USER directive.

PASS

The password used to log into the database. It is the same as the environment variable **DBI_PASS**. If a password is not needed, just don't set the PASS directive.

COLUMN_DEF

A comma−separated set of lines in the form NAME=TYPE(N), where NAME is the name of the field/column, TYPE is the SQL data type reference, and N is the length (if needed). Most Interchange fields should be the fixed−length character type, something like char(128). In fact, this is the default if a type is not chosen for a column. There can be as many lines as needed. This is not a DBI parameter, it is specific to Interchange.

NAME

A space−separated field of column names for a table. Normally not used. Interchange should resolve the column names properly upon query. Set this if a catalog errors out with "dbi: can't find field names" or the like. The first field should always be **code**. This is not a DBI parameter, it is specific to Interchange. All columns must be listed, in order of their position in the table.

NUMERIC

Tells Interchange not to quote values for this field. It allows numeric data types for SQL databases. It is placed as a comma−separated field of column names for a table, in no particular order. This should be defined if a numeric value is used because many DBD drivers do not yet support type queries.

UPPERCASE

Tells Interchange to force field names to UPPER case for row accesses using the `[item−data ...]`, `[loop−data ...]`, `[item−field ...`, etc. Typically used for Oracle and some other SQL implementations.

DELIMITER

A Interchange delimiter type, either TAB,CSV,PIPE,%%,LINE or the corresponding numeric type. The default for SQL databases is TAB. Use DELIMITER if another type will be used to import. This is not a DBI parameter. It is specific to Interchange.

KEY

The keying default of `code` in the first column of the database can be changed with the KEY directive. Don't use this unless prepared to alter all searches, imports, and exports accordingly. It is best to just accept the default and make the first column the key for any Interchange database.

ChopBlanks, LongReadLen, LongTruncOK, RaiseError, etc.

Sets the corresponding DBI attribute. Of particular interest is `ChopBlanks`, which should be set on drivers which by default return space−padded fixed−length character fields (Solid is an example).
The supported list as of this release of Interchange is:

```
ChopBlanks
CompatMode
LongReadLen
```

```
LongTruncOk
PrintError
RaiseError
Warn
```

Issue the shell command `perldoc DBI` for more information.

Here is an example of a completely set up DBI database on MySQL, using a comma−separated value input, setting the DBI attribute `LongReadLen` to retrieve an entire field, and changing some field definitions from the default char(128):

```
Database   products  products.csv  dbi:mysql:minivend
Database   products  USER          minivend
Database   products  PASS          nevairbe
Database   products  DELIMITER     CSV

# Set a DBI attribute
Database   products  LongReadLen   128

# change some fields from the default field type of char(128)
# Only applies if Interchange is importing from ASCII file
# If you set a field to a numeric type, you must set the
# NUMERIC attribute
Database   products  COLUMN_DEF    "code=char(20) NOT NUL primary key"
Database   products  COLUMN_DEF    price=float, discount=float
Database   products  COLUMN_DEF    author=char(40), title=char(64)
Database   products  COLUMN_DEF    nontaxable=char(3)
Database   products  NUMERIC       price
Database   products  NUMERIC       discount
```

MySQL, DBI, and DBD::mysql must be completely installed and tested, and have created the database `minivend`, for this to work. Permissions are difficult on MySQL. if having trouble, try starting the MySQL daemon with `safe_mysqld −−skip−grant−tables &` for testing purposes.

To change to ODBC, the only changes required might be:

```
Database products  DSN         dbi:ODBC:TCP/IP localhost 1313
Database products  ChopBlanks  1
```

The DSN setting is specific to a ODBC setup. The `ChopBlanks` setting takes care of the space−padding in Solid and some other databases. It is not specific to ODBC. Once again, DBI, DBD::ODBC, and the appropriate ODBC driver must be installed and tested.

46.2. SQL Access Methods

An Interchange SQL database can be accessed with the same tags as any of the other databases can. Arbitrary SQL queries can be passed with the `[query sql="SQL STATEMENT"]` MML tag.

46.3. Importing from an ASCII File

When importing a file for SQL, Interchange by default uses the first column of the ASCII file as the primary key, with a `char(16)` type, and assigns all other columns a `char (128)` definition. These definitions can be changed by placing the proper definitions in COLUMN_DEF Database directive attribute:

```
Database  products  COLUMN_DEF  price=char(20), nontaxable=char(3)
```

This can be set as many times as desired, if it will not fit on the line.

```
Database  products  COLUMN_DEF  price=char(20), nontaxable=char(3)
Database  products  COLUMN_DEF  description=char(254)
```

To create an index automatically, append the information when the value is in quotes:

```
Database  products  COLUMN_DEF  "code=char(14) primary key"
```

The field delimiter to use is TAB by default, but can be changed with the Database DELIMITER directive:

```
Database  products products.csv dbi:mSQL:minivend:localhost:1114
Database  products DELIMITER  CSV
```

To create other secondary keys to speed sorts and searches, do so in the COLUMN_DEF:

```
Database  products COLUMN_DEF  "author=char(64) secondary key"
```

Or use external database tools. NOTE: Not all SQL databases use the same index commands.

To use an existing SQL database instead of importing, set the NoImport directive in catalog.cfg to include any database identifiers not to be imported:

```
   NoImport  products inventory
```

WARNING: If Interchange has write permission on the products database, be careful to set the NoImport directive or create the proper .sql file. If that is not done, and the database source file is changed, the SQL database could be overwritten. In any case, always back up the database before enabling it for use by Interchange.

---

47. Managing DBM Databases

47.1. Making the Database

The DBM databases can be built offline with the `offline` command. The directory to be used for output is specified either on the command line with the `-d` option, or is taken from the `catalog.cfg` directive `OfflineDir` -- `offline` in the catalog directory by default. The directory must exist. The source ASCII files should be present in that directory, and the DBM files are created there. Existing files will be overwritten.

```
offline -c catalog [-d offline_dir]
```

Do a `perldoc VENDROOT/bin/offline` for full documentation.

47.2. Updating Individual Records

If it takes a long time to build a very large DBM database, consider using the `bin/update` script to change just one field in a record, or to add from a corrections list.

The database is specified with the `-n` option, or is 'products' by default.

The following updates the products database `price` field for item 19-202 with the new value 25.00:

```
update -c catalog -f price 25.00
```

More than one field can be updated on a single command line.

```
update -c catalog -f price -f comment 25.00 "That pitchfork couple"
```

The following takes input from `file`, which must be formatted exactly like the original database, and adds/corrects any records contained therein.

```
update -c catalog -i file
```

Invoke the command without any arguments for a usage message describing the options.

48. The Search Engine

Interchange implements a search engine which will search the product database (or any other file) for items based on customer input. It uses either forms or link−based searches that are called with the special page name `scan`. The search engine uses many special Interchange tags and variables.

If the search is implemented in a link or a form, it will always display formatted results on the results page, an Interchange page that uses some combination of the `[search−region]`, `[search−list]`, `[more−list]`, `[more]`, and other Interchange tags to format and display the results. The search results are usually a series of product codes/SKUs or other database keys, which are then iterated over similar to the `[item−list]`.

Note: Examples of search forms and result pages are included in the demos.

Two search engine interfaces are provided, and five types of searching are available. The default is a text−based search of the first products database source file (i.e., products.txt). A binary search of a dictionary−ordered file can be specified. An optional Glimpse search is enabled by placing the command specification for Glimpse in the `catalog.cfg` directive `Glimpse`. There is a range−based search, used in combination with one of the above. And finally, there is a fully−coordinated search with grouping.

The default, a text based search, sequentially scans the lines in the target file. By default it returns the first field (delineated by the delimiter for that database, for every line matching the search specification. This corresponds to the product code, which is then used to key specific accesses to the database.

The text−based search is capable of sophisticated field−specific searches with fully−independent case−sensitivity, substring, and negated matching.

48.1. The Search Form

A number of variables can be set on search forms to determine which search will be used, what fields in the database it will search, and what search behavior will be.

Here is a simple search form:

```
<FORM ACTION="[area search]" METHOD=POST>
<INPUT TYPE="text" SIZE="30" NAME="mv_searchspec">
<INPUT TYPE="submit" VALUE="Search">
</FORM>
```

When the "Search" submit button is pressed (or <ENTER> is pressed), Interchange will search the `products.txt` file for the string entered into the text field `mv_searchspec`, and return the product code pertaining to that line.

The same search for a fixed string, say "shirt," could be performed with the use of a hot link, using the special scan URL:

```
[page search="se=shirt"]See our shirt collection![/page]
```

The default is to search every field on the line. To match on the string "shirt" in the product database field "description," modify the search:

```
<INPUT TYPE="hidden" NAME="mv_search_field" VALUE="description">
```

In the hot−linked URL search:

```
[page search="
              se=shirt
              sf=category
          "]See our shirt collection![/page]
```

To let the user decide on the search parameters, use checkboxes or radiobox fields to set the fields:

```
Search by author
   <INPUT TYPE="checkbox" NAME="mv_search_field" VALUE="author">
Search by title
    <INPUT TYPE="checkbox" NAME="mv_search_field" VALUE="title">
```

Fields can be stacked. If more than one is checked, all checked fields will be searched.

48.2. Glimpse

To use the Glimpse search, the Glimpse index must be built based on files in the ProductDir, or wherever the files to be searched will be located. If Interchange was installed in the default `/usr/local/lib/minivend`, the command line to build the index for the products file would be:

```
chdir products
glimpseindex −b −H . products.txt
```

There are several ways to improve search speed for large catalogs. One method that works well for large `products.txt` files is to split the `products.txt` file into small index files (in the example, 100 lines) with the split(1) UNIX/POSIX command. Then, index it with Glimpse:

```
split −100 products.txt index.txt.
glimpseindex −H /usr/local/lib/minivend/products index.txt.*
```

This will dramatically increase search speeds for large catalogs, at least if the search term is relatively unique. If it is a common string, in a category search, for example, it is better to use the text−based search.

To search for numbers, add the `−n` option to the Glimpse command line.

Note: A large catalog is one of more than several thousand items; smaller ones have acceptable speed in any of the search modes.

If the Glimpse executable is not found at Interchange startup, the Glimpse search will be disabled and the regular text−based search used instead.

There are several things to watch for while using Glimpse, and a liberal dose of the Glimpse documentation is suggested. In particular, the spelling error capability will not work in combination with the field−specific search. Glimpse selects the line, but Interchange's text−based search routines disqualify it when checking to see if the search string is within one of the specified fields.

To use field−specific searching on Glimpse, tell it what the field names are. If the search is on the products database (file), nothing is needed for the default is to use the field names from the products database. If it is

some other field layout, specify the file to get the field names from with `mv_field_file` (ff).

48.3. Fast Binary Search

Fast binary searching is useful for scanning large databases for strings that match the beginning of a line. They use the standard Perl module Search::Dict, and are enabled through use of the `mv_dict_look`, `mv_dict_end`, `mv_dict_limit`, `mv_dict_fold`, and `mv_dict_order` variables.

The field to search is the first field in the file, the product code should be in the second field, delimited by TAB. Set the `mv_return_fields=1` to return the product code in the search.

The search must be done on a dictionary−ordered pre−built index, which can be produced with the database INDEX modifier. See Dictionary indexing with INDEX.

If using the `mv_dict_look` parameter by itself, and the proper index file is present, Interchange will set the options:

```
mv_return_fields=1
mv_dict_limit=−1
```

This will make the search behave much like the simple search described above, except it will be much faster on large files and will match only from the beginning of the field. Here is an example. A `title` index has been built by including in `catalog.cfg`:

```
Database   products   INDEX    title
```

Note: The ASCII source file must be "touched" to rebuild the index and the database.

Now, specify in a form:

```
<FORM ACTION="[process href=search]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_dict_limit VALUE=title>
<INPUT NAME=mv_dict_look>
</FORM>
```

or in a URL:

```
[page search="dl=Van Gogh/di=title"]
```

This search is case−sensitive. To do the same thing case−insensitively:

```
Database   products   INDEX    title:f

<FORM ACTION="[process href=search]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_dict_limit VALUE=title>
<INPUT TYPE=hidden NAME=mv_dict_fold  VALUE=1>
<INPUT NAME=mv_dict_look>
</FORM>

[page search="dl=Van Gogh/di=title/df=1"]
```

48.4. Coordinated and Joined Searching

Interchange will do a complete range of tests on individual columns in the database. To use this function, set `mv_coordinate` to Yes (co=yes in the one−click syntax). In order to use coordinated searching, the number of search fields must equal the number of search strings.

To make sure that is the case, use the `mv_search_map` variable. It allows variables to be mapped to others in the search specification. For example:

```
<INPUT TYPE=hidden NAME=mv_search_map VALUE="
    mv_searchspec=search1
    mv_searchspec=search2
    mv_searchspec=search3
    ">
<INPUT TYPE=hidden NAME=mv_search_field VALUE=title>
<INPUT TYPE=hidden NAME=mv_search_field VALUE=artist>
<INPUT TYPE=hidden NAME=mv_search_field VALUE=category>
Artist: <INPUT NAME=search1 VALUE="">
Title:  <INPUT NAME=search2 VALUE="">
Genre:  <INPUT NAME=search3 VALUE="">
```

Even if the user leaves one blank, the search will work.

Leading/trailing whitespace is stripped from all lines in the `mv_search_map` variable, so it can be positioned as shown for convenience.

Coordinated searches may be joined with the output of another table if set one of the `mv_search_field` values is set to a `table:column` pair. Note that this will slow down large searches considerably unless there is another search specification, as the database must be accessed for every search line If there is a search field that qualifies for a regular expression search function, or conducting a binary search with `mv_dict_look`, or are not doing an `OR` search, the penalty should not be too great as only matching lines will cause an access to the database.

Individual field operations can then be specified with the `mv_column_op` (or op) parameter. The operations include:

```
operation            string     numeric   equivalent
---------
equal to               eq         ==          =
not equal              ne         !=          <>
greater than           gt         >
less than              lt         <
less than/equal to     le         <=
greater than/equal to  ge         >=
regular expression     rm                      =~ , LIKE
regular expression NOT rn                      !~
exact match            em
```

An example:

```
[page scan
        co=yes
        sf=title
        se=Sunflowers
        op=em
```

```
        sf=artist
        se=Van Gogh
        op=rm          ] Sunflowers, Van Gogh [/page]

[page search="
        co=yes

        sf=title
        se=Sunflowers
        nu=0
        op=!~

        sf=artist
        se=Van Gogh
        op=rm
        nu=0

        sf=inventory:qty
        se=1
        op=>=
        nu=1
    "] Any in stock except Sunflowers, Van Gogh [/page]
```

Note that in the second example, nu=0 must be specified even though that is the default. This is to set the proper correspondence. To avoid having to do this, use Interchange's option array feature:

```
[page search.0="
                sf=title
                se=Sunflowers
                op=!~
            "
      search.1="
                sf=artist
                se=Van Gogh
            "
      search.2="
                sf=inventory:qty
                se=1
                op=>=
                nu=1
            "
    ] Any in stock except Sunflowers, Van Gogh [/page]
```

The `co=yes` is assumed when specifying a multiple search.

The second search will check the stock status of the painting provided there is an `inventory` table as in some of the Interchange demo catalogs. If the `qty` field is greater than or equal to 1, the product will be picked. If out of stock, it will not be found.

It always helps to have an `rm` type included in the search. This is used to pre−screen records so that database accesses only need be made for already−matching entries. If accesses must be made for every record, large searches can get quite slow.

48.5. Specifying a Text−Based Search with SQL−Like Syntax

If Jochen Wiedmann's `SQL::Statement` module is installed, a SQL syntax can be specified for the text−based search. (This is not the same as the SQL search, treated below separately. It would work on an SQL table, but only on the ASCII text source file, not on the actual database.)

This syntax allows this form setup:

```
Artist: <INPUT NAME="artist">
Title:  <INPUT NAME="title">
<INPUT TYPE=hidden NAME="mv_sql_query"
        VALUE="
            SELECT code FROM products
            WHERE artist LIKE artist
            AND    title LIKE title">
```

If the right hand side of an expression looks like a column, i.e., is not quoted, the appropriate form variable is substituted. (If used in a one−click, the corresponding scratch variable is used instead.) The assumption is reversed for the left−hand side. If it is a quoted string, the column name is read from the passed values. Otherwise, the column name is literal.

```
Search for: <INPUT NAME="searchstring"><BR>
Search in   <INPUT TYPE="radio" NAME="column" VALUE="title"> title
    <INPUT TYPE="radio" NAME="column" VALUE="artist"> artist
    <INPUT TYPE=hidden NAME="mv_sql_query"
      VALUE="SELECT code FROM products WHERE 'column' LIKE searchstring">
```

Once again, this does not conduct a search on an SQL database, but formats a corresponding text−based search. Parentheses will have no effect, and an OR condition will cause all conditions to be OR. The searches above would be similar to:

```
[page search="
            co=yes
            sf=artist
            op=rm
            se=[value artist]
            sf=title
            op=rm
            se=[value title]
        "  ]
    Search for [value artist], [value title]
[/page]

[page search="
            co=yes
            sf=[value column]
            op=rm
            se=[value searchstring]
        "  ]
Search for [value searchstring]
       in  [value column]
[/page]
```

48.6. Range Searching

Range searching allows qualification of search returns with a field that must be within a certain numeric or alphanumeric range. To use it, set the `mv_range_look` variable to the products database field, or a column/field number for another file. Then, set the corresponding `mv_range_min` and `mv_range_max`

variables with a selectable field.

```
<INPUT TYPE="hidden" NAME="mv_range_look" VALUE="price">
    Search on Price
Min <SELECT NAME="mv_range_min">
         <OPTION value=0 SELECTED> Free
         <OPTION value=1000000> $1,000,000
         <OPTION value=10000000> $10,000,000
         <OPTION value=20000000> $20,000,000
         <OPTION value=40000000> $40,000,000
    </SELECT><BR>
Max <SELECT NAME="mv_range_max">
        <OPTION value=0 SELECTED> no object
        <OPTION value=1000000> $1,000,000
        <OPTION value=10000000> $10,000,000
        <OPTION value=20000000> $20,000,000
        <OPTION value=40000000> $40,000,000
    </SELECT>
```

The value of 0 for `mv_range_max` is equivalent to infinity if doing a numeric search. This makes it impossible to search for a ceiling of 0 with a negative `mv_range_min`.

The fields are stackable, so more than one range to check can be set. The order is significant, in the sense that the array of field names and minimum/maximum values must be kept in order to achieve correspondence.

The optional `mv_range_alpha` specification allows alphanumeric range matching for the corresponding field. If it is set, and the fields are stacked, they must all be set. The `mv_case` field does apply if it is set. Otherwise, the comparison is without regard to case.

If ONLY a range search is required, all lines with `mv_return_all`=yes must be selected in order to make the search operate. Range−only searches will be quite slow for large databases since every line must be scanned. It should be quite usable for catalogs of less than 10,000 items in size on a fast machine. Using it in combination with another search technique (in the same query) will yield faster search returns.

48.7. One−Click Searches

Interchange allows a search to be passed in a URL, as shown above. Just specify the search with the special page parameter search or special page `scan`. Here is an example:

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

This is the same:

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

Here is the same thing from a home page (assuming /cgi−bin/vlink is the CGI path for Interchange's vlink):

```
<A HREF="/cgi−bin/vlink/scan/se=Impressionists/sf=category">
```

```
    Impressionist Paintings
</A>
```

The two−letter abbreviations are mapped with these letters:

```
ac  mv_all_chars
bd  mv_base_directory
bs  mv_begin_string
co  mv_coordinate
cs  mv_case
cv  mv_verbatim_columns
de  mv_dict_end
df  mv_dict_fold
di  mv_dict_limit
dl  mv_dict_look
DL  mv_raw_dict_look
do  mv_dict_order
dr  mv_record_delim
em  mv_exact_match
er  mv_spelling_errors
ff  mv_field_file
fi  mv_search_file
fm  mv_first_match
fn  mv_field_names
hs  mv_head_skip
ix  mv_index_delim
lb  mv_search_label
lo  mv_list_only
lr  mv_search_line_return
md  mv_more_decade
ml  mv_matchlimit
mm  mv_max_matches
MM  mv_more_matches
mp  mv_profile
ms  mv_min_string
ne  mv_negate
ng  mv_negate
np  mv_nextpage
nu  mv_numeric
op  mv_column_op
os  mv_orsearch
ra  mv_return_all
rd  mv_return_delim
rf  mv_return_fields
rg  mv_range_alpha
rl  mv_range_look
rm  mv_range_min
rn  mv_return_file_name
rr  mv_return_reference
rs  mv_return_spec
rx  mv_range_max
SE  mv_raw_searchspec
se  mv_searchspec
sf  mv_search_field
sg  mv_search_group
si  mv_search_immediate
sp  mv_search_page
sq  mv_sql_query
sr  mv_search_relate
st  mv_searchtype
su  mv_substring_match
```

```
tc  mv_sort_command
td  mv_table_cell
tf  mv_sort_field
th  mv_table_header
to  mv_sort_option
tr  mv_table_row
un  mv_unique
va  mv_value
```

These can be treated just the same as form variables on the page, except that they can't contain a new line. If using the multi−line method of specification, the characters will automatically be escaped for a URL.

IMPORTANT NOTE: An incompatibility in earlier Interchange catalogs is specifying `[page scan/se=searchstring]`. This is interpreted by the parser as `[page scan/se="searchstring"]` and will cause a bad URL. Change this to `[page scan se=searchstring]`, or perhaps better yet:

```
[page search="
                se=searchstring
        "]
```

A one−click search may be specified in three different ways.

Original

To do an OR search on the fields category and artist for the strings "Surreal" and "Gogh," while matching substrings, do:

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

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

Multi−Line

Specify parameters one to a line, as well.

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

Any "unsafe" characters will be escaped. To search for trailing spaces (unlikely), quote.

Ampersand

Substitute & for / in the specification and be able to use / and quotes and spaces in the specification.

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

Any "unsafe" characters will be escaped.

48.8. Setting Display Options with mv_value

A value can be specified that will be set in the link with the `mv_value` parameter. It takes an argument of `var=value`, just as setting a normal variable in an Interchange profile. Actually `mv_value` is a misnomer, it will almost never be used in a form where variable values can be set. Always specify it in a one−click search with `va=var=value`. Example:

```
[page href=scan
      arg="se=Renaissance
           se=Impressionists
           va=category_name=Renaissance and Impressionist Paintings
           os=yes"]Renaissance and Impressionist Paintings[/page]
```

Display the appropriate category on the search results page with `[value category_name]`.

48.9. In−Page Searches

To specify a search inside a page with the `[search−region parameters*]` tag. The parameters are the same as the one−click search, and the output is always a newline−separated list of the return objects, by default, a series of item codes.

The `[loop ...]` tag directly accepts a search parameter. To search for all products in the categories "Americana" and "Contemporary," do:

```
[loop search="
    se=Americana
    se=Contemporary
    os=yes
    sf=category9
    "]
Artist: [loop−field artist]<BR>
Title: [loop−field title]<P>
[/loop]
```

The advantage of the in−page search is that searches can be embedded within searches, and there can be straight unchanging links from static HTML pages.

To place an in−page search with the full range of display in a normal results page, use the `[search−region]` tag the same as above, except that `[search−list]`, `[more−list]`, and `[more]` tags can be placed within it. Use them to display and format the results, including paging. For example:

```
[search−region  more=1
                search="
                     se=Americana
                     sf=category
                     ml=2
                "]
[more−list][more][/more−list]
```

```
[search-list]
<A MV="page [item-code]" HREF="flypage.html">
    [item-field title]<A>, by [item-field artist]
[/search-list]
[no-match]
    Sorry, no matches for [value mv_searchspec].
[/no-match]
[/search-region]
```

To use the same page for search paging, make sure to set the `sp=page` parameter.

48.10. Search Profiles

An unlimited number of search profiles can be predefined that reside in a file or files. To use this, make up a series of lines like:

```
mv_search_field=artist
mv_search_field=category
mv_orsearch=yes
```

These correspond to the Interchange search variables that can be set on a form. Set it right on the page that contains the search.

```
[set artist_profile]
mv_search_field=artist
mv_search_field=category
mv_orsearch=yes
[/set]
```

This is the same:

```
[set artist_profile]
sf=artist
sf=category
os=yes
[/set]
```

Then, in the search form, set a variable with the name of the profile:

```
<INPUT TYPE=hidden NAME=mv_profile VALUE=artist_profile>
```

In a one-click search, use the `mp` modifier:

```
[page scan se=Leonardo/mp=artist_profile]A left-handed artist[/page]
```

They can also be placed in a file. Define the file name in the `SearchProfile` directive. The catalog must be reconfigured for Interchange to read it. The profile is named by placing a name following a __NAME__ pragma:

```
__NAME__ title_search
```

The __NAME__ must begin the line, and be followed by whitespace and the name.

The special variable `mv_last` stops interpretation of search variables. The following variables are always interpreted:

```
mv_dict_look
mv_searchspec
mv_range_look
mv_range_min
mv_range_max
```

Other than that, if `mv_last` is set in a search profile, and there are other variables on the search form, they will not be interpreted.

To place multiple search profiles in the same file, separate them with __END__, which must be on a line by itself.

48.11. Search Reference

The supplied `simple/srchform.html` and `simple/results.html` pages show example search forms. Modify them to present the search in any way desired. Be careful to use the proper variable names for passing to Interchange. It is also necessary to copy the hidden variables as−is. They are required to interpret the request as a search.

Note: The following definitions frequently refer to field name and column and column number. All are the references to the columns of a searched text file as separated by delimiter characters.

The field names can be specified in several ways.

ProductFiles

If the file to be searched is left empty in the search form or definition (it is set with `mv_search_file (fi)`), the text files associated with the products databases will be searched, and field names are already available as named in the first line of the file(s). This is defined to be `products.txt` in the Interchange demonstrations.
Be Careful if using SQL! If the database is change and not exporte with `[tag export products][/tag]`, searches will not be successful.

Other database files

If the file or files to be searched are ASCII delimited files, and have field names specified on the first line of the file, Interchange will read the first line (of the first file) and determine the field names.

Other files

If the file or files to be searched are ASCII delimited files, but don't have field names specified on the first line of the file, set the variable `mv_field_names` to a comma−separated list of field names as they will be referenced.

Fields can also always be specified by an integer column number, with 0 as the first column.

mv_all_chars

Scan abbreviation: ac=[1|0]. Set this if searching is anticipated for lots of punctuation characters that might be special characters for Perl. The characters ()[]\$^ are included.

mv_base_directory

Scan abbreviation: bd=/directory/name. In the text search, set to the directory from which to base file searches. File names without leading / characters will be based from there. In the Glimpse search, passed to Glimpse with the `-H` option, and Glimpse will look for its indices there. Default is ProductDir.
If an absolute path directory is used, for security enable it in the users session with:

```
[set /directory/name]1[/set]
```

This prevents users from setting an arbitrary value and viewing arbitrary files.

mv_begin_string

If this is set, the string will only match if it is at the beginning of a field. The handling is a bit different for the default AND search compared to the OR search. With OR searches all words are searched for from the beginning of the field, with AND searches all are.
This is a multiple parameter. If `mv_coordinate` is in force, it should be set as many times as necessary to match the field/searchstring combination. If set only once, it applies to all fields. If set more than once but not as many times as the fields, it will default to off.

mv_case

If this item is set to `No`, the search will return items without regard to upper or lower case. This is the default. Set to `Yes` if case should be matched. Implement with a checkbox <INPUT TYPE=CHECKBOX> field.
If stacked to match the `mv_search_field` and `mv_searchspec` variables, and `mv_coordinate` is set, it will operate only for the corresponding field.

mv_coordinate

> If this item is set to `Yes`, and the number of search fields equals the number of search specs, the search will return only items that match field to spec. (The search specifications are set by stacked `mv_searchspec` and `mv_search_field` variables.)
> Case sensitivity, substring matching, and negation all work on a field−by field basis according to the following:

If only one instance of the option is set, it will affect all fields.
If the number of instances of the option is greater than or equal to the number of search specs, all will be used independently. Trailing instances will be ignored.
If more than one instance of the options are set, but fewer than the number of search specifications, the default setting will be used for the trailing unset options.
If a search specification is blank, it will be removed and all case−sensitivity/negation/substring options will be adjusted accordingly.

mv_dict_end

If the string at the beginning of a line lexically exceeds this value, matching will stop. Ignored without `mv_dict_look`.

mv_dict_fold

Make dictionary matching case−insensitive. Ignored without `mv_dict_look`.

Note: This is the reverse sense from `mv_case`.

---

mv_dict_limit

Automatically set the limiting string (mv_dict_end) to be one character greater than the mv_dict_look variable, at the character position specified. A value of 1, for instance, will set the limiting string to "fprsythe" if the value of `mv_dict_look` is "forsythe". A useful value is −1, which will increment the last character (setting the mv_dict_end to "forsythf" in our example). This prevents having to scan the whole file once a unique match is found.

Note: The order of this and the `mv_dict_end` variable is significant. Each will overwrite the other.

If this is set to a non−numeric value, an automatic mode is entered which looks for a dictionary−indexed file that corresponds to the file name plus `.field`, where `field` is whatever `mv_dict_limit` is set to. The actual value of mv_dict_limit is set to `−1`. If the file does not exist, the original file is silently used. Also, the value of `mv_return_fields` is set to `1` to correspond to the location of the key in the auto−indexed file. To illustrate:

```
<INPUT TYPE=hidden NAME=mv_dict_limit  VALUE=category>
<INPUT TYPE=hidden NAME=mv_search_file VALUE="products.txt">
```

is equal to:

```
<INPUT TYPE=hidden NAME=mv_dict_limit    VALUE="−1">
<INPUT TYPE=hidden NAME=mv_search_file   VALUE="products.txt.category">
<INPUT TYPE=hidden NAME=mv_return_fields VALUE="1">
```

The real utility would be in a form construct like

```
Search for
<SELECT NAME=mv_dict_limit>
<OPTION> author
<OPTION> title
</SELECT> beginning with <INPUT NAME=mv_dictlook>
```

which would allow automatic binary search file selection.
Combined with the `INDEX` attribute to the Database directive, this allows fast binary search qualification combined with regular `mv_searchspec` text searches.

mv_dict_look

The string at which to begin matching at in a dictionary−based search. If not set, the `mv_dict_end`, `mv_dict_fold`, and `mv_dict_case` variables will be ignored. May be set in a search profile based on other form variables.

mv_dict_order

Make dictionary matching follow dictionary order, where only word characters and whitespace matter. Ignored without `mv_dict_look`.

mv_doit

This must be set to `search` to make this a search page.

mv_exact_match

Normally Interchange searches match words, as opposed to sentences. This behavior can be overridden with `mv_exact_match`, which when set will place quotes around any value in `mv_searchspec` or `mv_dict_look`.

mv_field_names

Deprecated in favor of in−list sorting. Defines the field names for the file being searched. This guarantees that they will be available, and prevents a disk access if using named fields on a search file (that is not the product database ASCII source, where field names are already known). This must be exactly correct, or it will result in anomalous search operation. Usually passed in a hidden field or search profile as a comma−separated list.

Note: Use this on the product database only if planning on both pre−sorting with `mv_sort_field` and then post−sorting `with [sort]field:opt[/sort]`.

---

mv_first_match

Normally Interchange will return the first page of a search. If this variable is set, it will start the search return at the match specified, even if there is only one page. If set to a value greater than the number of matches, it will act as if no matches were found.

mv_head_skip

Normally Interchange searches all lines of an index/product file but the first. Set this to the number of lines to skip at the beginning of the index. Default is 1 for the text search, which skips the header line in the product file. Default is 0 for a Glimpse search.

mv_index_delim

Sets the delimiter for counting fields in a search index. The default is TAB.

mv_matchlimit

The page size for matches that are returned. If more matches than mv_matchlimit are found, the search paging mechanism will be employed if the proper `[more−list]` is present. Can be implemented as a scrolling list (INPUT TYPE=SELECT) or radiobox (INPUT TYPE=RADIO) field.

mv_max_matches

The maximum number of records that will be returned in a search. Default is 2000. This only applies to Glimpse. Use `mv_matchlimit` to set the search page size.

mv_min_string

Sets the minimum size of a search string for a search operation. Default is 4 for the Glimpse search, and 1 for

the text search.

mv_negate

Specifies that records NOT matching the search criteria will be returned. Default is no. It is not operative for the Glimpse search.
If stacked to match the `mv_search_field` and `mv_searchspec` variables, and `mv_coordinate` is set, it will operate only for the corresponding field.

mv_orsearch

If this item is set to `Yes`, the search will return items matching any of the words in `searchspec`. The default is No.

mv_profile

Selects one of the pre−defined search specifications set by the `SearchProfile` directive. If the special variable within that file, `mv_last`, is defined, it will prevent the scanning of the form input for further search modifications. The values of `mv_searchspec` and `mv_dict_look` are always scanned, so specify this to do the equivalent of setting multiple checkboxes or radioboxes with one click, while still reading the search input text.

mv_range_alpha

Sets the type of match, numeric or alphanumeric, for the range search in its corresponding range field. The search will return true, assuming it is greater than the `mv_range_min` specification, if the field searched is less than or equal to `mv_range_max`, in an alphanumeric sense.

mv_range_look

This sets a field to scan for a range of numbers. It must be accompanied with corresponding `mv_range_min` and `mv_range_max` variables. It can be specified with either a field name or a column number.

mv_range_max

Sets the high bound for the range search in its corresponding range field. The search will return true, assuming it is greater than the `mv_range_min` specification, if the field searched is less than or equal to `mv_range_max`. To set the bound at infinity, or whatever your integer limit is, set `mv_range_min` to 0.

mv_range_min

Sets the low bound for the range search in its corresponding range field. The search will return true, assuming it is less than the `mv_range_max` specification, if the field searched is less than or equal to `mv_range_min`.

mv_record_delim

Sets the delimiter for counting records in a search index. The default is newline, which works for the products and most line−based index files.

mv_return_fields

The fields that should be returned by the match, specified either by field name or by column number. Specify 0 as the first field to be returned if searching the products database, since that is the key for accessing database fields.

mv_return_spec

Returns the string specified as the search (i.e., the value of `mv_searchspec`) as the one and only match. Typically used in a SKU/part number search.

mv_search_field

The field(s) to be searched, specified either by column name or by column number.
If the number of instances matches the number of fields specified in the `mv_searchspec` variable and `mv_coordinate` is set to true, each search field (in order specified on the form) will be matched with each search spec (again in that order).

mv_search_file

In the text search, set this variable to the file(s) to be scanned for a match. The default, if not set, is to scan the default ProductFiles (i.e., products.txt). If set multiple times in a form (for a text search), will cause a search all the files. One file name per instance.
In the Glimpse search, follows the Glimpse wildcard−based file name matching scheme. Use with caution and a liberal dose of the Glimpse man page.

mv_search_match_count

Set by the search to indicate the total number of matches found.

mv_search_over_msg

The message that should be displayed if there is an overflow condition (`mv_matchlimit` is exceeded). Overrides the SearchOverMsg directive. It is cleared by Interchange if there is no overflow. Somewhat deprecated by match paging.

mv_search_page

The Interchange−style name of the page that should display the search results. Overrides the `FrameSearchPage` directive, and the default value of `search`.

mv_searchspec

The actual search string that is typed in by the customer. It is a text INPUT TYPE=TEXT field, or can be put in a select (drop−down) list to enable category searches. If multiple instances are found, they will be concatenated just as if multiple words had been placed in a text field.
The user can place quotes around words to specify that they match as a string. To enable this by default, use the `mv_exact_match` variable.
If `mv_dict_look` has a value, and `mv_searchspec` does not, then `mv_searchspec` will be set to the value of `mv_dict_look`.
If the number of instances matches the number of fields specified in the `mv_search_field` variable and `mv_coordinate` is set to true, each search field (in order specified on the form) will be matched with each search spec (again in that order).

mv_searchtype

If set to Glimpse, selects the Glimpse search (if Glimpse is defined).
If set to db, iterates over every row of the database (not the associated text source file).
If set to sql, same as `db`.
If set to text, selects the text−based search.
When using `st=db`, returned keys may be affected by `TableRestrict`. See `CATALOG.CFG`.
Defaults to text if Glimpse is not defined; defaults to Glimpse if it is defined. This can allow use of both search types if that is desirable. For instance, searching for very common strings is better done by the text−based search. An example might be searching for categories of items instead of individual items.

mv_sort_field

The file field(s) the search is to be sorted on, specified in one of two ways. If the file(s) to be searched have a header line (the first line) that contains delimiter−separated field names, it can be specified by field name. It can also be specified by column number (the code or key is specified with a value of 0, for both types). These can be stacked if coming from a form or placed in a single specification separated by commas.

Note: If specifying a sort for the product database, `mv_field_names` must be specified if doing a fieldname−addressed post−sort.

---

mv_sort_option

The way that each field should be sorted. The flags are `r`, `n`, and `f`, reverse, numeric, and case−insensitive respectively. These can be stacked if coming from a form or placed in a single specification separated by commas. The stacked options will be applied to the sort fields as they are defined, presuming those are stacked.

mv_spelling_errors

The number of spelling errors that will be tolerated. Ignored unless using Glimpse. For a large catalog, limit this to two.

mv_substring_match

If `mv_substring_match` is set to `Yes`, matches on substrings as well as whole words. Typically set this for dictionary−based searches.
If stacked to match the `mv_search_field` and `mv_searchspec` variables and `mv_coordinate` is set, it will operate only for the corresponding field.

mv_unique

If set to a true value, causes the sort to return only unique results. This operates on whatever the search return is, as defined by `mv_return_fields`.

mv_value

This is normally only used in the one−click search (va=var=value). It allows setting of a session variable based on the clicked link, which makes for easy definition of headers and other display choices. (If had trouble using `mv_searchspec` for this before, this is what is needed.)

48.12. The Results Page

Once a search has been completed, there needs to be a way of presenting the output. By default, the `SpecialPage` search is used. It is set to `results` in the distribution demo, but any number of search pages can be specified by passing the value in the search form specified in the variable `mv_search_page`.

On the search page, some special Interchange tags are used to format the otherwise standard HTML. Each of the iterative tags is applied to every code returned from the search. This is normally the product code, but could be a key to any of the arbitrary databases. The value placed by the `[item-code]` tag is set to the first field returned from the search.

The basic structure looks like this:

```
[search-region]
[search-list]
    your iterating code, once for each match
[/search-list]
[no-match]
    Text / tags to be output if no matches found (optional but recommended)
[/no-match]
[more-list]
    More / paging area (optional)
[/more-list]
[/search-region]
```

TIP FOR MV3 PORTS: A `[search-list][/search-list]` must always be surrounded by a `[search-region][/search-region]` pair. This is a change from Interchange 3.

[search-list]

Starts the representation of a search list. Interchange tags can be embedded in the search list, yielding a table or formatted list of items with part number, description, price, and hyperlinks to order or go to its catalog page.
The example tags shown have an `item-` prefix, which is the default. Set any prefix desired with the `prefix` parameter to `[search-region]`:

```
[search-region prefix=my]
[search-list]
    SKU:   [my-code]
    Title: [my-data products title]
[/search-list]
[/search-region]
```

The standard set of Interchange iterative ITL tags are available. They are interpolated in this order:

```
[item-alternate N] true [else] false [/else] [/item-alternate]
[if-item-param named_field] true [else] false [/else] [/if-item-param]
[item-param named_field]
[if-item-pos N] true [else] false [/else] [/if-item-pos]
[item-pos N]
[if-item-field products_field] true [else] false [/else] [/if-item-field]
[item-field products_column]
[item-increment]
[item-accessories]
[item-code]
```

```
[item-description]
[if-item-data table column] true [else] false [/else] [/if-item-data]
[item-data table column]
[item-price N* noformat=1*]
[item-calc] [/item-calc]
[item-change marker]
    [condition]variable text[/condition]
    true
    [else] false [/else]
[/item-change marker]
[item-last] condition [/item-last]
[item-next] condition [/item-next]
```

Note: those that reference the shopping cart do not apply, i.e., [item−quantity], [item−modifier ...] and friends.

[/search−list]

Ends the search list.

[no−match]

Starts the region of the search results page that should be returned if there is no match (and no error) for the search. If this is not on the page, the special page nomatch will be displayed instead.

[/no−match]

Ends the no match region.

[sort database:field:option* database:field:option*]

Sorts the search list return based on database fields. If no options are supplied, sorts according to the return code. See SORTING.
This is slow, and it is far better to pre−sort the return in the search specification.

[item−change marker]

Active only within `[search−list][/search−list]`.
Along with the companion `[/item−change marker]`, surrounds a region which should only be output when a field (or other repeating value) changes its value. This allows indented lists similar to database reports to be easily formatted. The repeating value must be a tag interpolated in the search process, such as `[item−field field]` or `[item−data database field]`.
Of course, this will only work as expected when the search results are properly sorted.
The `marker` field is mandatory, and is also arbitrary, meaning that any marker can be selected as long as it matches the marker associated with `[/item−change marker]`. The value to be tested is contained within a `[condition]value[/condition]` tag pair. The `[item−change marker]` tag also processes an `[else] [/else]` pair for output when the value does not change. The tags may be nested as long as the markers are different.
The following is a simple example for a search list that has a field `category` and `subcategory` associated with each item:

```
<TABLE>
<TR><TH>Category</TH><TH>Subcategory</TH><TH>Product</TH></TR>
[search-list]
```

```
<TR>
   <TD>
        [item-change cat]

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

                [item-field category]
        [else]
                 
        [/else]
        [/item-change cat]
   </TD>
   <TD>
        [item-change subcat]

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

                [item-field subcategory]
        [else]
                 
        [/else]
        [/item-change subcat]
   </TD>
   <TD> [item-field name] </TD>
[/search-list]
</TABLE>
```

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

[/item−change marker]

Companion to `[item−change marker]`.

[matches]

Replaced with the range of match numbers displayed by the search page. Looks something like "1−50". Make sure to insert this item between a `[more−list]` and `[/more−list]` element pair.

[more−list next_img* prev_img* page_img* border* border_current*]

Starts the section of the search page which is only displayed if there are more matches than specified in `mv_matchlimit`. If there are less matches than the number in mv_matchlimit, all text/html between the `[more_list]` and `[/more_list]` elements is stripped.
Use in conjunction with the `[more]` element to place pointers to additional pages of matches.
If the optional arguments `next_img`, `prev_img`, and/or `page_img` are present, they represent image files that will be inserted instead of the standard 'Next,' 'Previous,' and page number. If `prev_img` is `none`, then no previous link will be output. If `page_img` is `none`, then no links to pages of matches will be output. These are URLs, are substituted for with *ImageDir* and friends, and will be encased in IMG tags. Lastly, `border` is the border number to put.
In addition, if `page_img` is used, it will be passed an argument of the digit that is to be represented. This would allow an image generator program to be used, generating page numbers on the fly. The `border` and `border_selected` values are integers indicating the border that should be put around images in the `page_img` selection. The <border_selected> is used for the current page if set.
\Examples:

`[more-list next.gif prev.gif page_num.cgi 3]` causes anchors of:

```
Previous   <IMG SRC="prev.gif" Border=3>
Page 1     <IMG SRC="/cgi-bin/page_num.cgi?1">
Page 2     <IMG SRC="/cgi-bin/page_num.cgi?2">
Next       <IMG SRC="next.gif" Border=3>
```

`[more-list next.gif prev.gif page_num.cgi]` causes anchors of:

```
Previous   <IMG SRC="prev.gif">
Page 1     <IMG SRC="/cgi-bin/page_num.cgi?1">
Page 2     <IMG SRC="/cgi-bin/page_num.cgi?2">
Next       <IMG SRC="next.gif">
```

`[more-list next.gif prev.gif 0 0]` causes anchors of:

```
Previous   <IMG SRC="prev.gif" Border=0>
Page 1     <IMG SRC="/cgi-bin/page_num.cgi?1">
Page 2     <IMG SRC="/cgi-bin/page_num.cgi?2">
Next       <IMG SRC="next.gif" Border=0>
```

To set custom text for the "Previous" and "Next" usually used, define the `next_img`, `prev_img`, and `page_img` with `[next-anchor][/next-anchor]`, `[prev-anchor][/prev-anchor]` and `[page-anchor][/page-anchor]`. The string $PAGE$ will be replaced with the page number in the latter. The same example:

```
[more-list 0 0 0]
[next-anchor] Forward [/next-anchor]
[prev-anchor] Back [/prev-anchor]
[page-anchor] Page $PAGE$ [/page-anchor]
[more]
[/more-list]
```

will display `Forward Page 1 Page 2 Back` for 2 pages.
As shown, pass a 0 for the arguments of each to tell Interchange to look for the assignments.
If have many pages of matches and don't wish to have all displayed at once, set `[decade-next][/decade-next]` and `[decade-prev][/decade-prev]`. If set them empty, a search with 31 pages will display pages 21-30 like:

```
Previous 1 2 3 4 5 6 7 8 9 10 [more>>] Next
```

and pages 11-20 like:

```
Previous [<<more] 11 12 13 14 15 16 17 18 19 20 [more>>] Next
```

If set to `[decade-next](higher)[/decade-next]` and `[decade-prev](lower)[/decade-prev]`, the following will be displayed:

```
Previous (lower) 11 12 13 14 15 16 17 18 19 20 (higher) Next
```

Of course, image-based anchors can be used as well.

[/more-list]

Companion to `[more-list]`.

[more]

Inserts a series of hyperlinks that will call up the next matches in a series. They look like this:

```
Previous 1 2 3 4 5 6 Next
```

The current page will not be a hyperlink. Every time the new link is pressed, the list is re−built to correspond to the current page. If there is no `Next` or `Previous` page, that link will not be shown.
See the `fr_resul.html` or `search.html` files for examples. Make sure to insert this item between a `[more−list]` and `[/more−list]` element pair.

[process−search]

Calls the search with the proper URL, including Interchange session tags. Used as the ACTION value for the search form.

[process−target frame]

Calls the search with the proper URL, including Interchange session tags. Used as the ACTION value for the search form if the results are to be targeted to a different window than the one set by SearchFrame (which is "_self" by default).

49. Sorting

Interchange has standard sorting options for sorting the search lists, loop lists, and item lists based on the contents of database fields. In addition, it adds list slices for limiting the displayed entries based on a start value and chunk size (or start and end value, from which a chunk size is determined). All accept a standard format sort tag which must be directly after the list call:

```
[loop 4 3 2 1]
[sort -2 +2]
    [loop-code]
[/loop]

[search-list]
[sort products:category:f]
    [item-price] [item-description]<BR>
[/search-list]

[item-list]
[sort products:price:rn]
    [item-price] [item-code]<BR>
[/item-list]

[loop search="ra=yes"]
[sort products:category products:title]
[loop-field category] [loop-field title] <BR>
[/loop]
```

All sort situations, `[search list]`, `[loop list]`, `[tag each table]`, and `[item-list]`, take options of the form:

```
[sort database:field:option* -n +n =n-n ... ]
```

database

The Interchange database identifier. This must be supplied and should normally be 'products' if using the default name for the database.

field

The field (column) of the database to be sorted on.

option

None, any, or combinations of the options:

```
f   case-insensitive sort (folded) (mutually exclusive of n)
n   numeric order (mutually exclusive of f)
r   reverse sort
```

-n

The starting point of the list to be displayed, beginning at 1 for the first entry.

+n

The number of entries to display in this list segment.

=n−n

The starting and ending point of the list display. This is an alternative to −n and +n. They should be specified in only one form. If both are specified, the last one will take effect.

...

Don't really put `...` in. This means that many sort levels are specified. Lots of sort levels with large databases will be quite slow.

Multiple levels of sort are supported, and database boundaries on different sort levels can be crossed. Cross−database sorts on the same level are not supported. If using multiple product databases, they must be sorted with embedded Perl. This is actually a feature in some cases, all items in a `used` database can be displayed before or after new ones in `products`.

Examples, all based on the `simple` demo:

Loop list

```
[loop 00-0011 19-202 34-101 99-102]
[sort products:title]
    [loop-code] [loop-field title]<BR>
[/loop]
```

Will display:

```
34-101 Family Portrait
00-0011 Mona Lisa
19-202 Radioactive Cats
99-102 The Art Store T-Shirt
```

\Alternatively:

```
[loop 00-0011 19-202 34-101 99-102]
[sort products:title -3 +2]
    [loop-code] [loop-field title]<BR>
[/loop]
```

\Displays:

```
19-202 Radioactive Cats
99-102 The Art Store T-Shirt
```

The tag `[sort products:title =3-4]` is equivalent to the above.

Search list

A search of all products (i.e., http://yoursystem.com/cgi−bin/simple/scan/ra=yes):

```
[search-list]
[sort products:artist products:title:rf]
    [item-field artist] [item-field title]<BR>
```

```
[/search-list]
```

will display:

```
Gilded Frame
Grant Wood American Gothic
Jean Langan Family Portrait
Leonardo Da Vinci Mona Lisa
Salvador Dali Persistence of Memory
Sandy Skoglund Radioactive Cats
The Art Store The Art Store T-Shirt
Vincent Van Gogh The Starry Night
Vincent Van Gogh Sunflowers
```

Note the reversed order of the title for Van Gogh and the presence of the accessory item Gilded Frame at the front of the list. It has no artist field and, as such, sorts first).
Adding a slice option:

```
[search-list]
[sort products:artist products:title:rf =6-10]
    [item-field artist] [item-field title]<BR>
[/search-list]
```

will display:

```
Sandy Skoglund Radioactive Cats
The Art Store The Art Store T-Shirt
Vincent Van Gogh The Starry Night
Vincent Van Gogh Sunflowers
```

If the end value/chunk size exceeds the size of the list, only the elements that exist will be displayed, starting from the start value.

Shopping cart

```
[item-list]
[sort products:price:rn]
    [item-price] [item-code]<BR>
[/item-list]
```

will display the items in the shopping cart sorted on their price, with the most expensive shown first. NOTE: This is based on the database field and doesn't take quantity price breaks or discounts into effect. Modifier values or quantities cannot be sorted.

Complete database contents

```
[tag each products]
[sort products:category products:title]
[loop-field category] [loop-field title] <BR>
[/tag]
```

A two level sort that will sort products based first on their category, then on their title within the category.

Note that large lists may take some time to sort. If a product database contains many thousands of items, using the `[tag each products]` sort is not recommended unless planning on caching or statically building

pages.

50. Shipping

Interchange has a powerful custom shipping facility that performs UPS and other shipper lookups, as well as a flexible rule−based facility for figuring cost by other methods.

50.1. Shipping Cost Database

The shipping cost database (located in ProductDir/shipping.asc) is a tab−separated ASCII file with six fields: code, text description, criteria (quantity or weight, for example), minimum number, maximum number, and cost. None of the fields are case−sensitive.

To define the shipping database in a catalog configuration file, set the Variable `MV_SHIPPING` to what would be its contents.

To set the file to be something other than `shipping.asc` in the products directory, set the `Special` directive:

```
Special  shipping.asc  /home/user/somewhere/shipping_defs
```

There are two styles of setting which can be mixed in the same file. The first is line−based and expects six or more TAB−separated fields. They would look like:

```
default No shipping weight  0   99999999    0

upsg    UPS Ground  weight  0   0   e Nothing to ship!
upsg    UPS Ground  weight  0   150 u Ground [default zip 98366] 3.00
upsg    UPS Ground  weight  150 999999  e @@TOTAL@@ lbs too heavy for UPS
```

The second is a freeform method with a `mode: Description text` introducing the mode line. The special encoding is called out by indented parameters. The below is identical to the above:

```
upsg: UPS Ground
    criteria    weight
    min         0
    max         0
    cost        e Nothing to ship!

    min         0
    max         150
    cost        u
    table       2ndDayAir
    geo         zip
    default_geo 98366
    adder       3

    min         150
    max         999999
    cost        e @@TOTAL@@ lbs too heavy for UPS
```

The second format has several advantages. Multiple lines can be spanned with the <<HERE document format, like so:

```
upsg: UPS Ground
    criteria    <<EOF
```

```
[perl]
    return 'weight' if $Values->{country} eq 'US';
    return 'weight' if ! $Values->{country};
    # Return blank, don't want UPS
    return '';
[/perl]
EOF
```

The definable fields are, in order, for the tab−separated format:

MODE

The unique identifier for that shipping method. It may be repeated as many times as needed.

DESCRIPTION

Text to describe the method (can be accessed on a page with the `[shipping−description]` element).

CRITERIA

Whether shipping is based on weight, quantity, price, etc. Valid Interchange tags can be placed in the field to do a dynamic lookup. If a number is returned, that is used as the accumulated criteria. That is, the total of weight, quantity, or price as applied to all items in the shopping cart.
See `Criteria Determination` below.

MINIMUM

The low bound of quantity/weight/criteria this entry applies to.

MAXIMUM

The high bound of quantity/weight/criteria this entry applies to. The first found entry is used in case of ties.

COST

The method of developing cost. It can be a number which will be used directly as the shipping cost, or a function, determined by a single character at the beginning of the field:

```
f       Formula (MML tags OK, evaluated as Perl)
x       Multiplied by a number
[uA−Z]  UPS−style lookup
m       Interchange chained cost lookup (all items summed together)
i       Interchange chained cost lookup (items summed individually)
```

NEXT

The `next` field supplies an alternative shipping mode to substitute if the cost of the current one is zero.

ZONE

The UPS zone that is being defined.

QUERY

Interchange tags which will return a SQL query to select lines matching this specification. The current mode is replaced with this selection. If there is a query parameter of ?, it will be replaced with the mode name.

QUAL

The geographic qualification (if any) for this mode.

PERL

Perl code that is read and determines the criterion, not the cost. Use the `cost` option with "f" as the prelim to supply Perl code to determine cost.

TOTAL

Set to the accumulated criterion before passing to Perl.

OPT

Used to maintain UPS and freeform options. Normally these are set by separate lines in the shipping definition.

50.2. Criteria Determination

The criteria field varies according to whether it is the first field in the shipping file exactly matching the mode identifier. In that case, it is called the main criterion. If it is in subsidiary shipping lines matching the mode (with optional appended digits), it is called a qualifying criterion. The difference is that the main criterion returns the basis for the calculation (i.e., weight or quantity), while the qualifying criterion determines whether the individual line may match the conditions.

The return must be one of:

quantity

The literal value quantity as the main criterion will simply count the number of items in the shopping cart and return it as the accumulated criteria. If using a database table field named `quantity`, use the `table::field` notation.

o <field name> or <table>::<field name>

A valid database field (column) name as main criterion will cause the number of items in the shopping cart to be multiplied by the value of the field for each item to obtain the accumulated criteria. If the table is not supplied, defaults to the first `ProductFiles` table.

o n.nn

Where **n.nn** is any number, it will be directly used as the accumulated criteria. This can be effectively returned from a Perl subroutine or Interchange `[calc][item-list] ... [/item-list][/calc]` to create custom shipping routines.

IMPORTANT NOTE: The above only applies to the first field that matches the shipping mode exactly. Following criteria fields contain qualifier matching strings.

50.3. Shipping Calculation Modes

There are eight ways that shipping cost may be calculated. The method used depends on the first character of the `cost` field in the shipping database.

N.NN (digits)

If the first character is a digit, a number is assumed and read directly as the shipping cost.

e

If the first character is an `e`, a cost of zero is returned and an error message is placed in the session value `ship_message` (i.e., `[data session ship_message]` or $Session->{ship_message}).

f

If the character `f` is the first, Interchange will first interpret the text for any Interchange tags and then interpret the result as a formula. It is read as Perl code; the entire set of Interchange objects may be referenced with the code.

i

Specifies a chained shipping lookup which will be applied to each item in the shopping cart.

m

Specifies a chained shipping lookup which will be applied to the entire shopping cart.

u

Calls the UPS-style lookup. Can pre-define as many as desired. Though if want to do the hundreds available, it is best done on-the-fly.

x

If an `x` is first, a number is expected and is applied as a fixed multiplier for the accumulated criterion (@@TOTAL@@).

A-Z

If the first character is a capital letter, calls one of the 26 secondary UPS-style lookup zones. (Deprecated now that zones can be named directly).

50.4. How Shipping is Calculated

1. The base code is selected by reading the value of `mv_shipmode` in the user session. If it has not

been explicitly set, either by means of the DefaultShipping directive or by setting the variable on a form (or in an order profile), it will be `default`.
The mv_shipmode must be in the character class [A−Za−z0−9_]. If there are spaces, commas, or nulls in the value, they will be read as multiple shipping modes.

2. The modes are selected from the d
   The criterion field is found. If it is quantity, it is the total quantity of items on the order form. If it is any other name, the criterion is calculated by multiplying the return value from the product database field for each item in the shopping cart, multiplied by its quantity. If the lookup fails due to the column or row not existing, a zero cost will be returned and an error is sent to the catalog error log. If a number is returned from an Interchange tag, that number is used directly.
   Entries in the shipping database that begin with the same string as the shipping mode are examined. If none is found, a zero cost is returned and an error is sent to the catalog error log.

Note: The same mode name may be used for all lines in the same group, but the first one will contain the main criteria.

---

1. The value of the accumulated criteria is examined. If it falls within the minimum and maximum, the cost is applied.
2. If the cost is fixed, it is simply added.
3. If the cost field begins with an `x`, the cost is multiplied by the accumulated criterion, i.e., price, weight, etc.
4. If the cost field begins with `f`, the formula following is applied. Use @@TOTAL@@ as the value of the accumulated criterion.
5. If the cost field begins with `u` or a single letter from A−Z, a UPS−style lookup is done.
6. If the cost field begins with `s`, a Perl subroutine call is made.
7. If the cost field begins with `e`, zero cost is returned and an error placed in the session **ship_message** field, available as `[data session ship_message]`.

Here is an example shipping file using all of the methods of determining shipping cost.

Note: The columns are lined up for reading convenience. The actual entries should have **one** tab between fields.

```
global Option   n/a               0   0      g PriceDivide

rpsg   RPS      quantity          0   0      R RPS products/rps.csv
rpsg   RPS      quantity          0   5      7.00
rpsg   RPS      quantity          6   10     10.00
rpsg   RPS      quantity          11  150    x .95

usps   US Post  price             0   0      0
usps   US Post  price             0   50     f 7 + (1 * @@TOTAL@@ / 10)
usps   US Post  price             50  100    f 12 + (.90 * @@TOTAL@@ / 10)
usps   US Post  price             100 99999  f @@TOTAL@@ * .05

upsg   UPS      weight [value state]  0   0    e Nothing to ship.
upsg   UPS      AK HI             0   150    u upsg [default zip 980] 12.00 round
upsg   UPS                        0   150    u Ground [default zip 980] 2.00 round
upsg   UPS                        150 9999   e @@TOTAL@@ lb too heavy for UPS

upsca  UPS/CA   weight            0   0      c C UPS_Canada products/can.csv
upsca  UPS/CA   weight            −1   −1    o PriceDivide=0
upsca  UPS/CA   weight            0   150    C upsca [default zip A7G] 5.00
```

```
upsca  UPS/CA   weight           150 99999  e @@TOTAL@@ lb too heavy for UPS
```

global

This is a global option setting, called out by the `g` at the beginning. PriceDivide tells the shipping routines to multiply all shipping settings by the PriceDivide factor, except those explicitly set differently with the `o` individual modifier. This allows currency conversion. (Currently the only option is PriceDivide.)

rpsg

If the user selected RPS, (code rpsg) and the quantity on the order was 3, the cost of 7.00 from the second rpsg line would be applied. If the quantity were 7, the next entry from the third rpsg line would be selected for a cost of 10.00. If the quantity were 15, the last rpsg would be selected and the quantity of 15 multiplied by 0.95, for a total cost of 14.25.

usps

The next mode, `usps`, is a more complicated formula using price as the criteria. If the total price of all items in the shopping cart (same as `[subtotal]` without quantity price breaks in place) is from 1 to 50, the cost will be 7.00 plus 10 percent of the order. If the total is from 50.01 to 100, the cost will be 12.00 plus 9 percent of the order total. If the cost is 100.01 or greater, 5 percent of the order total will be used as the shipping cost.

upsg

The next, `upsg`, is a special case. It specifies a UPS lookup based on the store's UPS zone and two required values (and two optional arguments):

1. Weight
2. The zip/postal code of the recipient of which only the first three digits are used.
3. A fixed amount to add to the cost found in the UPS tables (use 0 as a placeholder if specifying roundup)
4. If set to 'round,' will round the cost up to the next integer monetary unit.

If the cost returned is zero, the reason will be placed as an error message in the session variable ship_message (available as `[data session ship_message]`).
UPS weights are always rounded up if any fraction is present.
The routines use standard UPS lookup tables. First, the UPS Zone file must be present. That is a standard UPS document specific to the retailer's area that must be obtained from UPS. It is entered into and made available to Interchange in TAB−delimited format. (As of March 1997, use the standard .csv file distributed by UPS on their Web site at `www.ups.com`.) Specify it with the UpsZoneFile directive. It is usually named something like `NNN.csv`, where NNN is the first three digits of the originating zip code. If placed in the products directory, the directive would look like:

```
UPSZoneFile  products/450.csv
```

Second, obtain the cost tables from UPS (again, get them from `www.ups.com`) and place them into an Interchange database. That database, its identifier specified with the first argument (Ground in the example) of the cost specification, is consulted to determine the UPS cost for that weight and rate schedule.
In the example below, use a database specification like:

```
Database  Ground  Ground.csv  CSV
```

A simple shipping cost qualification can be appended to a UPS lookup. If any additional parameters are present after the five usual ones used for UPS lookup, they will be interpreted as a Perl subroutine call. The syntax is the same as if it was encased in the tag `[perl sub] [/perl]`, but the following substitutions are made prior to the call:

```
@@COST@@  is replaced with whatever the UPS lookup returned
@@GEO@@   is replaced with the zip (or other geo code)
@@ADDER@@ is replaced with the defined adder
@@TYPE@@  is replaced with the UPS shipping type
@@TOTAL@@ is replaced with the total weight
```

The example above also illustrates geographic qualification. If the value of the form variable state on the checkout form is AK or HI, the U.S. states Alaska and Hawaii, a $10.00 additional charge (over and above the normal $2.00 handling charge) is made. This can also be used to select on country, product type, or any other qualification that can be encoded in the file.

upsca

The next entry is just like the UPS definition except it defines a different lookup zone file (`products/can.csv`) and uses a different database, `upsca`. It also disables the global PriceDivide option for itself only, not allowing currency conversion. Otherwise, the process is the same.
Up to 27 different lookup zones can be defined in the same fashion. If one of the cost lines (the last field) in the `shipping.asc` file begins with a `c`, it configures another lookup zone which must be lettered from `A to Z`. It takes the format:

```
c X name file* length* multiplier*
```

where X is the letter from `A−Z`. The name is used internally as an identifier and must be present. The optional `file` is relative to the catalog root (like `UpsZoneFile` is). If it is not present, the file equal to `name` in the products directory (ProductDir) will be used as the zone file. If the optional digit `length` is present, that determines the number of significant digits in the passed postal/geo code.
When the optional `multiplier` is present, the weight is multiplied by it before doing the table lookup. This allows shipping weights in pounds or kilograms to be adapted to a table using the opposite as the key. Remember, the match on weight must be exact, and Interchange rounds the weight up to the next even unit.
To define the exact equivalent of the UPS lookup zone, do the following:

```
c U UPS products/450.csv 3 1
```

The only difference is that the beginning code to call the lookup is upper−case `U` instead of lower−case `u`.

50.5. More On UPS−Style Lookup

The UPS−style lookup uses two files for its purposes, both of which need to be in a format like UPS distributes for US shippers.

The zone file is a file that is usually specific to the originating location. For US shippers shipping to US locations, it is named for the first three digits of the originating zip code with a CSV extension. For example, `450.csv`.

It has a format similar to:

```
low − high, zone,zone,zone,zone
```

The `low` entry is the low bound of the geographic location; `high` is the high bound. (By geographic location, the zip code is meant.) If the first digits of the zip code, compared alphanumerically, fall between the low and high values, that zone is used as the column name for a lookup in the rate database. The weight is used as the row key.

The first operative row of the zone file (one without leading quotes) is used to determine the zone column name. In the US, it looks something like:

Dest. ZIP,Ground,3 Day Select,2nd Day Air,2nd Day Air A.M.,Next Day Air Saver,Next Day Air

Interchange strips all non−alpha characters and comes up with:

DestZIP,Ground,3DaySelect,2ndDayAir,2ndDayAirAM,NextDayAirSaver,NextDayAir

Therefore, the zone column (shipping type) that would be used for UPS ground would be "Ground," and that is what the database should be named. To support the above, use a `shipping.asc` line that reads:

```
upsg  UPS Ground  weight  0  150  u Ground [default zip 983]
```

and a `catalog.cfg` database callout of:

```
Database  Ground  Ground.csv  CSV
```

These column names can be changed as long as they correspond to the `identifier` of the rate database.

The rate database is a standard Interchange database. For U.S. shippers, UPS distributes their rates in a fairly standard comma−separated value format, with weight being the first (or key) column and the remainder of the columns corresponding to the zone which was obtained from the lookup in the zone file.

To adapt other shipper zone files to Interchange's lookup, they will need to fit the UPS US format. (Most of the UPS international files don't follow the U.S. format). For example, the 1998 Ohio−US to Canada file begins:

```
Canada Standard Zone Charts from Ohio
Locate the zone by cross−referencing the first three
characters of the destination Postal Code in the Postal
Range column.

Postal Range  Zone
A0A  A9Z      54
B0A  B9Z      54
C0A  C9Z      54
E0A  E9Z      54
G0A  G0A      51
G0B  G0L      54
G0M  G0S      51
G0T  G0W      54
```

It will need to be changed to:

```
Destination,canstnd
A0A−A9Z, 54
B0A−B9Z, 54
C0A−C9Z, 54
E0A−E9Z, 54
```

```
G0A-G0A, 51
G0B-G0L, 54
G0M-G0S, 51
G0T-G0W, 54
```

Match it with a `canstnd` CSV database that looks like this:

```
Weight,51,52,53,54,55,56
1,7.00,7.05,7.10,11.40,11.45,11.50
2,7.55,7.65,7.75,11.95,12.05,12.10
3,8.10,8.15,8.40,12.60,12.70,12.85
4,8.65,8.70,9.00,13.20,13.30,13.55
5,9.20,9.25,9.75,13.85,13.85,14.20
6,9.70,9.85,10.35,14.45,14.50,14.90
7,10.25,10.40,11.10,15.15,15.15,15.70
8,10.80,10.95,11.70,15.70,15.75,16.35
9,11.35,11.55,12.30,16.40,16.45,17.20
```

It is called out in catalog.cfg with:

```
Database canstnd canstnd.csv CSV
```

With the above, a 4-pound shipment to postal code E5C 4TL would yield a cost of 13.20.

50.6. Geographic Qualification

If the return value in the main criterion includes whitespace, the remaining information in the field is used as a qualifier for the subsidiary shipping modes. This can be used to create geographic qualifications for shipping, as in:

```
upsg   UPS Ground   weight [value state]   0     0     e No items selected
upsg   UPS Ground   AK HI                  0     150   u Ground [value zip] 12.00
upsg   UPS Ground                          0     150   u Ground [value zip] 3.00
```

If `upsg` is the mode selected, the value of the user session variable `state` is examined to see if it matches the geographic qualification on a whole-word boundary. If it is `AK` or `HI`, UPS Ground with an adder of 12 will be selected. If it "falls through," UPS Ground with an adder of 3 will be selected.

50.7. Handling Charges

Additional handling charges can be defined in the shipping file by setting the form variable `mv_handling` to a space, comma, or null-separated set of valid shipping modes. The lookup and charges are created in the same fashion, and the additional charges are added to the order. (The user is responsible for displaying the charge on the order report or receipt with a `[shipping handling]` tag, or the like.) All of the shipping modes found in mv_handling will be applied. If multiple instances are found on a form, the accordingly null-separated values will all be applied. NOTE: This should not be done in an item-list unless the multiple setting of the variables is accounted for.

To only process a handling charge once, do the following:

```
[item-list]
[if-item-field very_heavy]
[perl values]
    return '' if $Values->{mv_handling} =~ /very_heavy/;
```

```
    return "<INPUT TYPE=hidden NAME=mv_handling VALUE=very_heavy>";
[/perl]
[/if-item-field]
[/item-list]
```

A non-blank/non-zero value in the database field will trigger Perl code which will only set `mv_handling` once.

50.8. Default Shipping Mode

If a default shipping mode other than `default` is desired, enter it into the `DefaultShipping` directive:

```
DefaultShipping     upsg
```

This will make the entry on the order form checked by default when the user starts the order process, if it is put in the form:

```
<INPUT TYPE=RADIO NAME=mv_shipmode VALUE=upsg [checked mv_shipmode upsg]>
```

To force a choice by the user, make `mv_shipmode` a required form variable (with RequiredFields or in an order profile) and set `DefaultShipping` to zero.

51. User Database

Interchange has a user database function which allows customers to save any pertinent values from their session. It also allows the setting of database or file access control lists for use in controlling access to pages and databases on a user−by−user basis.

The database field names in the user database correspond with the form variable names in the user session. If there is a column named `address`, when the user logs in the contents of that field will be placed in the form variable `address`, and will be availabe for display with `[value address]`. Similarly, the database value is available with `[data table=userdb column=address key=username]`.

The ASCII file for the database will not reflect changes unless the file is exported with `[tag export userdb][/tag]`. It is not advisable to edit the ASCII file, as it will overwrite the real data that is in the DBM table. User logins and changes would be lost. Note: This would not happen with SQL, but editing the ASCII file would have no effect. It is recommended that the NoImport configuration directive be set accordingly.

The field names to be used are not set in concrete. They may be changed with options. Fields may be added or subtracted at any time. Most users will choose to keep the default demo fields for simplicity sake, as they cover most common needs. As distributed in the demo, the fields are:

```
code
accounts
acl
address
address_book
b_address
b_city
b_country
b_name
b_nickname
b_phone
b_state
b_zip
carts
city
country
db_acl
email
email_copy
fax
fax_order
file_acl
mv_credit_card_exp_month
mv_credit_card_exp_year
mv_credit_card_info
mv_credit_card_type
mv_shipmode
name
order_numbers
p_nickname
password
phone_day
phone_night
preferences
s_nickname
```

```
state
time
zip
```

A few of those fields are special in naming, though all can be changed via an option. A couple of the fields are reserved for Interchange's use.

Note: If not running with PGP or other encryption for credit card numbers, which is never recommended, it is important that the mv_credit_card_info field be removed from the database.

The special database fields are:

```
accounts         Storage for billing accounts book
address_book     Storage for shipping address book
b_nickname       Nickname of current billing account
carts            Storage for shopping carts
p_nickname       Nickname for current preferences
preferences      Storage for preferences
s_nickname       Nickname for current shipping address
db_acl           Storage for database access control lists
file_acl         Storage for file access control lists
acl              Storage for simple integrated access control
```

If not defined, the corresponding capability is not available.

Note: The fields `accounts`, `address_book`, `carts`, and `preferences` should be defined as a BLOB type, if using SQL. This is also suggested for the acl fields if those lists could be large.

Reserved fields include:

```
code        The username (key for the database)
password    Password storage
time        Last time of login
```

51.1. The [userdb ...] Tag

Interchange provides a `[userdb ...]` tag to access the UserDB functions.

```
[userdb
       function=function_name
       username="username"*
       assign_username=1
       username_mask=REGEX*
       password="password"*
       verify="password"*
       oldpass="old password"*
       crypt="1|0"*
       shipping="fields for shipping save"
       billing="fields for billing save"
       preferences="fields for preferences save"
       ignore_case="1|0"*
       force_lower=1
       param1=value*
       param2=value*
       ...
       ]
```

* Optional

It is normally called in an `mv_click` or `mv_check` setting, as in:

```
[set Login]
mv_todo=return
mv_nextpage=welcome
[userdb function=login]
[/set]

<FORM ACTION="[process-target]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_click VALUE=Login>
Username <INPUT NAME=mv_username SIZE=10>
Password <INPUT NAME=mv_password SIZE=10>
</FORM>
```

There are several global parameters that apply to any use of the `userdb` functions. Most importantly, by default, the database table is set to be userdb. If another table name must be used, include a `database=table` parameter with any call to `userdb`. The global parameters (default in parens):

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

By default the system crypt() call will be used to compare the password. This is best for security, but the passwords in the user database will not be human readable.

If no critical information is kept and Interchange administration is not done via the `UserDB` capability, use the `UserDB` directive (described below) to set encryption off by default:

```
UserDB   default   crypt   0
```

Encryption can still be set on by passing `crypt=1` with any call to a `new_account`, `change_pass`, or `login` call.

51.2. Setting Defaults with the UserDB Directive

The `UserDB` directive provides a way to set defaults for the user database. For example, to save and recall the scratch variable `tickets` in the user database instead of the form variable `tickets`, set:

```
UserDB   default   scratch  tickets
```

That makes every call to `[userdb function=login]` equivalent to `[userdb function=login scratch=tickets]`.

To override that default for one call only, use `[userdb function=login scratch="passes"]`.

To log failed access authorizations, set the `UserDB` profile parameter `log_failed` true:

```
UserDB  default  log_failed 1
```

To disable logging of failed access authorizations (the default), set the `UserDB` profile parameter `log_failed` to 0:

```
UserDB  default  log_failed 0
```

The `UserDB` directive uses the same key−value pair settings as the `Locale` and `Route` directives. If there are more than one set of defaults, set them in a hash structure:

```
UserDB  crypt_case  <<EOF
{
    'scratch'        => 'tickets',
    'crypt'          => '1',
    'ignore_case'    => '0',
}
EOF

UserDB  default  <<EOF
{
    'scratch'        => 'tickets',
    'crypt'          => '1',
    'ignore_case'    => '1',
}
EOF
```

Note: The usual here−document caveats apply. The "EOF" must be on a line by itself with no leading/trailing whitespace.

The last one to be set becomes the default.

The option `profile` selects the set to use. For usernames and passwords to be case sensitive with no encryption, pass this call:

```
[userdb function=new_account profile=case_crypt]
```

The username and password will be stored as typed in, and the password will be encrypted in the database.

51.3. User Database Functions

The user database features are implemented as a series of functions attached to the `userdb` tag. The functions are:

login

Active parameters: username, password, crypt, pass_field, ignore_case
Log in to Interchange. By default, the username is contained in the form variable `mv_username` and the password in `mv_password`. If the login is successful, the session value `username` (`[data session username]`) will be set to the user name.
This will recall the values of all non−special fields in the user database and place them in their corresponding user form variables.
The `CookieLogin` directive (catalog.cfg) allows users to save their username/password in a cookie. Expiration time is set by `SaveExpire`, renewed every time they log in. To cause the cookie to be generated originally, the form variable `mv_cookie_password` or `mv_cookie_username` must be set in the login form. The former causes both username and password to be saved, the latter just the username.

logout

Log out of Interchange. No additional parameters are needed.

new_account

Active parameters: username, password, verify, assign_username, username_mask, ignore_case
Create a new account. It requires the `username`, `password`, and `verify` parameters, which are by default contained in the form variables `mv_username`, `mv_password`, `mv_verify` respectively.
If the `assign_username` parameter is set, `UserDB` will assign a sequential username. The `counter` parameter can be used to set the filename (must be absolute), or the default of CATALOG_DIR/etc/username.counter can be accepted. The first username will be "U0001" if the counter doesn't exist already.
The `ignore_case` parameter forces the username and password to lower case in the database, in effect rendering the username and password case−insensitive.
If `username_mask` is set to a valid Perl regular expression (without the surrounding / /), then any username containing a matching string will not be allowed for use. For example, to screen out order numbers from being used by a random user:

```
[userdb function=new_account
        username_mask="^[A-Z]*[0-9]"
        ]
```

The `CookieLogin` directive (catalog.cfg) allows users to save their username/password in a cookie. Expiration time is set by `SaveExpire`, renewed every time they log in. To cause the cookie to be generated originally, the form variable `mv_cookie_password` or `mv_cookie_username` must be set in the login form. The former causes both username and password to be saved, the latter just the username.
To automatically create an account for every order, set the following in the `OrderReport` file:

```
[userdb function=new_account
        username="[value mv_order_number]"
        password="[value zip]"
        verify="[value zip]"
        database="orders"
        ]
```

This would be coupled with a login form that asks for order number and zip code, thereupon allowing the display of the contents of a transaction database with (presumably updated) order status information or a shipping company tracking number.

change_pass

Active parameters: username, password, verify, oldpass
Change the password on the currently logged−in account. It requires the `username`, `password`, `verify`, and `oldpass` parameters, which are by default contained in the form variables `mv_username`, `mv_password`, `mv_verify`, `mv_password_old` respectively.

set_shipping

Active parameters: nickname, shipping, ship_field
Place an entry in the shipping Address book. For example:

```
[userdb function=set_shipping nickname=Dad]
```

See Address Book below.

get_shipping

Active parameters: nickname, shipping, ship_field
Recall an entry from the shipping Address book. For example:

```
[userdb function=get_shipping nickname=Dad]
```

See Address Book below.

get_shipping_names

Active parameters: ship_field
Gets the names of shipping address book entries and places them in the variable `address_book`. By default, it does not return the values. To have the values returned, set the parameter `show` to 1, as in:

```
[set name=shipping_nicknames
     interpolate=1]
  [userdb function=get_shipping_names show=1]
[/set]
```

set_billing

Active parameters: nickname, billing, bill_field
Place an entry in the billing accounts book. For example:

```
[userdb function=set_billing nickname=discover]
```

See Accounts Book below.

get_billing

Active parameters: nickname, billing, bill_field
Recall an entry from the billing accounts book. For example:

```
[userdb function=get_billing nickname=visa]
```

See Accounts Book below.

save

Saves all non−special form values that have columns in the user database.

set_cart

Save the contents of a shopping cart.

```
[userdb function=set_cart nickname=christmas]
```

See Carts below.

get_cart

Active parameters: nickname, carts_field, target
Recall a saved shopping cart.

```
[userdb function=get_cart nickname=mom_birthday]
```

Setting `target` saves to a different shopping cart than the default main cart. The `carts_field` controls the database field used for storage.

set_acl

Active parameters: location, acl_field, delete
Set a simple acl. For example:

```
[userdb function=set_acl location=cartcfg/editcart]
```

This allows the current user to access the page "cartcfg/editcart" if it is access−protected.
To delete access, do:

```
[userdb function=set_acl location=cartcfg/editcart delete=1]
```

To display the setting at the same time as setting, use the `show` attribute:

```
[userdb function=set_acl location=cartcf/editcart show=1]
```

check_acl

Active parameters: location, acl_field
Checks the simple access control listing for a location, returning 1 if allowed and the empty string if not allowed.

```
[if type=explicit
    compare="[userdb
                function=check_acl
                location=cartcfg/editcart]"
]
[page cartcfg/editcart]Edit your cart configuration[/page]
[/if]
```

set_file_acl, set_db_acl

Active parameters: location, mode, db_acl_field, file_acl_field, delete
Sets a complex access control value. Takes the form:

```
[userdb function=set_file_acl
        mode=rw
        location=products/inventory.txt]
```

where mode is any value to be checked with `check_file_acl`. As with the simple ACL, use delete=1 to delete the location entirely.

check_file_acl, check_db_acl

Active parameters: location, mode, db_acl_field, file_acl_field
Checks a complex access control value and returns a true/false (1/0) value. Takes the form:

```
[userdb function=check_db_acl
        mode=w
        location=inventory]
```

where mode is any value to be checked with `check_file_acl`. It will return true, if the mode string is contained within the entry for that location. For example:

```
[if type=explicit
    compare="[userdb
                function=check_db_acl
                mode=w
                location=inventory]"
]
[userdb function=set_acl location=cartcfg/edit_inventory]
[page cartcfg/edit_inventory]You may edit the inventory database[/page]
[else]
[userdb function=set_acl location=cartcfg/edit_inventory delete=1]
Sorry, you can't edit inventory.
[/if]
```

51.4. Address Book

Address_book is a shipping address book. The shipping address book saves information relevant to shipping the order. In its simplest form, this can be the only address book needed. By default these form values are included:

```
s_nickname
name
fname
lname
address
address1
address2
address3
city
state
zip
country
phone_day
mv_shipmode
```

The first field is always the name of the form variable that contains the key for the entry. The values are saved with the `[userdb function=set_shipping]` tag call, and are recalled with `[userdb function=get_shipping]`. A list of the keys available is kept in the form value `address_book`, suitable for iteration in an HTML select box or in a set of links.

To get the names of the addresses, use the `get_shipping_names` function:

```
[userdb function=get_shipping_names]
```

By default, they are placed in the variable `address_book`. Here is a little snippet that builds a select box:

```
<FORM ACTION="[process-target]" METHOD=POST>
[userdb function=get_shipping_names]
[if value address_book]
<SELECT NAME="s_nickname">
[loop arg="[value address_book]"] <OPTION> [loop-code] [/loop]
</SELECT>
<INPUT TYPE=submit NAME=mv_check VALUE="Recall Shipping">
</FORM>
```

The same principle works with accounts, carts, and preferences.

To restore a cart based on the above, put in an `mv_check` routine:

```
[set Recall Shipping]
mv_todo=return
mv_nextpage=ord/basket
[userdb function=get_shipping nickname="[value s_nickname]"]
[/set]
```

When the `mv_check` variable is encountered, the contents of the scratch variable `Recall Shipping` are processed and the shipping address information inserted into the user form values. This is destructive of any current values of those user session variables, of course.

To change the fields that are recalled or saved, use the `shipping` parameter:

```
[userdb function=get_shipping
        nickname=city_and_state
        shipping="city state"]
```

Only the values of the `city` and `state` variables will be replaced.

51.5. Accounts Book

The accounts book saves information relevant to billing the order. By default these form values are included:

```
b_nickname
b_name
b_fname
b_lname
b_address
b_address1
b_address2
b_address3
b_city
```

```
b_state
b_zip
b_country
b_phone
purchase_order
mv_credit_card_type
mv_credit_card_exp_month
mv_credit_card_exp_year
mv_credit_card_info
```

The values are saved with the `[userdb function=set_billing]` tag call, and are recalled with `[userdb function=get_billing]`. A list of the keys available is kept in the form value `accounts`, suitable for iteration in an HTML select box or in a set of links.

51.6. Preferences

Preferences are miscellaneous session information. They include, by default, the following fields:

```
email
fax
phone_night
fax_order
email_copy
```

The field `p_nickname` acts as a key to select the preference set. To change the values that are included with the `preferences` parameter:

```
[userdb function=set_preferences
        preferences="email_copy email fax_order fax"]
```

or in catalog.cfg:

```
UserDB default preferences "mail_list email fax_order music_genre"
```

51.7. Carts

The contents of shopping carts may be saved or recalled in much the same fashion. See the Simple demo application `ord/basket.html` page for an example.

51.8. Controlling Page Access With UserDB

Interchange can implement a simple access control scheme with the user database. Controlled pages must reside in a directory which has a file named `.access` that is zero bytes in length. (If it is more than 0 bytes, only the RemoteUser or MasterHost may access files in that directory.)

Set the following variables in `catalog.cfg`:

```
Variable   MV_USERDB_ACL_TABLE  userdb
Variable   MV_USERDB_ACL_COLUMN acl
```

The `MV_USERDB_ACL_TABLE` is the table which controls access, and likewise the `MV_USERDB_ACL_TABLE` names the column in that database which will be checked for authorization.

The database entry should contain the complete Interchange−style page name of the page to be allowed. It will not match substrings.

For example, if the user `flycat` followed this link:

```
<A HREF="[area cartcfg/master_edit]">Edit</A>
```

Access would be allowed if the contents of the userdb were:

```
code    acl
flycat  cartcfg/master_edit
```

and disallowed if it were:

```
code    acl
flycat  cartcfg/master_editor
```

Access can be enabled with:

```
[userdb function=set_acl location="cartcfg/master_edit"]
```

Access can be disallowed with:

```
[userdb function=set_acl
        delete=1
        location="cartcfg/master_edit"]
```

Of course, a pre−existing database with the ACL values will work as well. It need not be in the UserDB setup.

52. Tracking and Back−End Order Entry

Interchange allows the entry of orders into a system through one of several methods. The `AsciiBackend` capability allows submission of parameters to an external order entry script. Support for SQL allows the entry of orders directly into an SQL database. Orders can be written to an ASCII file. They can be formatted precisely for e−mail−based systems. The orders can be placed in a DBM file. Finally, embedded Perl allows completely flexible order entry, including real−time credit card verification and settlement.

52.1. ASCII Backup Order Tracking

If `AsciiTrack` is set to a legal file name (based in VendRoot unless it has a leading "/"). A copy of the order is saved and sent in an e−mail.

If the file name string begins with a pipe "|," a program will be run and the output "piped" to that program. This allows easy back−end entry of orders with an external program.

52.2. Database Tracking

Once the order report is processed, the order is complete. Therefore, it is the ideal place to put Interchange tags that make order entries in database tables.

A good model is to place a single record in a database summarizing the order and a series of lines that correspond to each line item in the order. This can be in the same database table. If the order number itself is the key for the summary, a line number can be appended to the order number to show each line of the order.

The following would summarize a sample order number S00001 for part number 00−0011 and 99−102:

```
code      order_number part_number  quantity   price    shipping  tax
S00001    S00001                    3          2010     12.72     100.50
S00001−1 S00001       00−0011      2          1000     UPS       yes
S00001−2 S00001       99−102       1          10       UPS       yes
```

Fields can be added where needed, perhaps with order status, shipping tracking number, address, customer number, or other information.

The above is accomplished with Interchange's `[import ....]` tag using the convenient `NOTES` format:

```
[set import_status]
[import table=orders type=LINE continue=NOTES]

code: [value mv_order_number]
order_number: [value mv_order_number]
quantity: [nitems]
price: [subtotal noformat=1]
shipping: [shipping noformat=1]
tax: [salestax noformat=1]

[/import]

[item−list]
[import table=orders type=LINE continue=NOTES]
```

```
code: [value mv_order_number]-[item-increment]
order_number: [value mv_order_number]
quantity: [item-quantity]
price: [item-price noformat=1]
shipping: [shipping-description]
tax: [if-item-field nontaxable]No[else]Yes[/else][/if]

[/import][/item-list]
```

52.3. Order Routing

Interchange can send order emails and perform custom credit card charges and/or logging for each item. The Route directive is used to control this behavior, along with the `mv_order_route` item attribute.

If no Route is in the catalog, Interchange uses a default "mail out the order and show a receipt" model.

Routes are established with the `Route` directive, which is similar to the `Locale` directive. Each route is like a locale, so that key-value pairs can be set. Here is an example setting:

```
Route  mail  pgp_key         0x67798115
Route  mail  email           orders@akopia.com
Route  mail  reply           service@akopia.com
Route  mail  encrypt         1
Route  mail  encrypt_program "/usr/bin/pgpe -fat -q -r %s"
Route  mail  report          etc/report_mail
```

Note: Values with whitespace in them must be quoted.

You can also set the route in a valid Perl hash reference string:

```
Route  mail <<EOR
{
     pgp_key         => '0x67798115',
     email           => 'orders@akopia.com',
     reply           => 'service@akopia.com',
     encrypt         => 1,
     encrypt_program => q{/usr/bin/gpg -e -a -r '%s' --batch},
     report          => 'etc/report_mail',
}
EOR
```

This route would be used whenever the *mail* route was called by one of the three possible methods:

route called from master route

Called via the `cascade` parameter from the master route. This is the way that most routes are called in Interchange's [the Foundation manpage] demo. These routes treat the order as a whole.

route set in item

An item in the shopping cart has `mail` as the value in the attribute `mv_order_route`. This method is item-specific to this item (or group of items in route `mail`).

route set in the form variable mv_order_route

By setting a value in the `mv_order_route` form variable, you can specify one or more routes to run. This is the deprecated method used in earlier Interchange 4.6.x and Minivend 4 routes. It will still work fine.

The last route that is defined is the `master` route, by convention named *main*. Besides setting the global behavior of the routing, it provides some defaults for other routes. For example, if `encrypt_program` is set there, then the same value will be the default for all routes. Most settings do not fall through.

The attributes that can be set are:

attach

Determines whether the order report should be attached to the main order report e−mail. This is useful if certain items must be printed separately from others, perhaps for FAX to a fulfillment house.
cascade
A list of routes which should be pushed on the stack of routes to run, *after all currently scheduled routes are done*. NOTE: cascades can cause endless loops, so only one setting is recommended, that being the main route.

commit

Perl code which should be performed on a route commit.

commit_tables

Tables that are to be pre−opened before running the Perl commit code.

counter

The location of a counter file which should be used instead of `OrderCounter` for this route. It will generate a different value for `mv_order_number` for the route. This is normally used to obtain unique order references for multi−vendor routing.

credit_card

Determines whether credit card encryption should be done for this order. Either this or `encrypt` should always be set.

dynamic_routes

If set in the the master manpage route, will cause the the RouteDatabase manpage to be checked for a route. If it exists, it will be read in and the database copy used instead of the static copy build at catalog configuration time. If set in a subsidiary route, that route will be ignored during catalog.cfg, and `dynamic_routes` must be active for it to be seen.

email

The email address(es) where the order should be sent. Set just like the `MailOrderTo` directive, which is also the default.

encrypt

Whether the entire order should be encrypted with the **encrypt_program**. If `credit_card` is set, the credit card will first be encrypted, then the entire order encrypted.

encrypt_program

The encryption program incantaton which should be used. Set identically to the `EncryptProgram` directive, except that %s will be replaced with the `pgp_key`. Default is empty.

errors_to

Sets the `Errors-To:` e-mail header so that bounced orders will go to the proper address. Default is the same as `MailOrderTo`.

expandable

If set in the the master manpage route, route settings will be expanded for ITL tags. No effect if the route is not the master.

extended

Extended route settings that take the form of an Interchange option list; normally a Perl hash reference that will be read. These settings always overwrite any that currently exist, regardless of the order in which they are specified. For example:

```
Route  main   extended    { email => 'milton@akopia.com' }
Route  main   email       papabear@minivend.com
```

The ultimate setting of `email` will be `milton@akopia.com`.

increment

Whether the order number should be incremented as a result of this result. Default is not to increment, as the order number should usually be the same for different routes within the same customer order.

individual_track

A directory where individual order tracking files will be placed. The file name will correspond to the value of `mv_order_number`. This can be useful for batching orders via download.

individual_track_ext

The extension that will be added to the file name for `individual_track`. Must contain a period (`.`), if that is desired.

```
individual_track_ext     .pgp
```

individual_track_mode

A number representing the final permission mode for the `individual_track` file. Usually expressed in octal:

```
individual_track_mode    0444
```

master

If set, this route becomes the master route for `supplant`, `dynamic_routes`, `errors_to`, and `expandable`, and supplies the setting for `receipt` and the `attach` report. Switching `master` in midstream is unlikely to be successful −− it should certainly be the first route in a cascade.

payment_mode

If this is set, enables a payment mode for the route. (Payment modes are also set in the `Route` directive.)

pgp_cc_key

The PGP/GPG key selector that is used to determine which public key is used for encryption of credit cards only. With PGP 5 and 6, see appropriate values by using the command `pgpk -l`. For GPG, use `gpg --list-keys`. Defaults to the value of the pgp_key manpage.

pgp_key

The PGP key selector that is used to determine which public key is used for encryption. If `pgp_cc_key` is set, that key will be used for credit card encryption instead of `pgp_key`. With PGP 5 and 6, see appropriate values by using the command `pgpk -l`. For GPG, use `gpg --list-keys`. Defaults to the value of the pgp_key manpage.

profile

The custom order profile which should be performed to check the order *prior* to actually running the route. If it fails, the route will not be performed. See `OrderProfile` and `mv_order_profile`.

receipt

The receipt page that should be used for this routing. This only applies if supplant is set for the route, and that normally would only be in the default route.

report

The report page that should be used for this routing. If `attach` is defined, the contents of the report will be placed in a MIME attachment in the main order report.

reply

The `Reply-To` header that should be set. Default is the same as `email`.
If there are only word characters (A−Za−z0−9 and underscore), it describes an Interchange variable name where the address can be found.

rollback

Perl code which should be performed on a route rollback.

rollback_tables

Tables that are to be pre−opened before running the Perl rollback code.

supplant

Whether the master route should supplant the main order report. If set, the `AsciiTrack` operation will use this route and the normal Interchange order e−mail sequence will not be performed. This is normally set in the master route.

track

The name of a file which should be used for tracking. If the `supplant` attribute is set, the normal order tracking will be used as well.

track_mode

A number representing the final permission mode for the `track` file. Usually expressed in octal:

```
track_mode    0444
```

transactions

A list of tables to put in transactions mode at the beginning of the route. Used to ensure that orders get rolled back if another route fails.
The *first* route to open a table must have this parameter, otherwise transactions will not work. If any route fails (except ones marked error_ok) then a rollback will be done on these tables. If all routes succeed, a commit will be performed at the end of all order routes.

Individual item routing causes all items labeled with that route to be placed in a special sub−cart that will be used for the order report. This means that the `[item−list] LIST [/item−list]` will only contain those items, allowing operations to be performed on subsets of the complete order. The `[subtotal]`, `[salestax]`, `[shipping]`, `[handling]`, and `[total−cost]` tags are also affected.

Here is an example of an order routing:

```
Route  HARD      pgp_key          0x67798115
Route  HARD      email            hardgoods@akopia.com
Route  HARD      reply            service@akopia.com
Route  HARD      encrypt          1
Route  HARD      report           etc/report_mail

Route  SOFT      email            ""
Route  SOFT      profile          create_download_link
Route  SOFT      empty            1

Route  mail      pgp_key          0x67798115
Route  mail      email            orders@akopia.com
Route  mail      reply            service@akopia.com
Route  mail      encrypt          1
Route  mail      report           etc/report_all

Route  user      error_ok         1
Route  user      email            email
Route  user      reply            service@akopia.com
Route  user      report           etc/user_copy
```

```
Route  log       empty           1
Route  log       report          etc/log_transaction
Route  log       transactions    "transactions orderline inventory"
Route  log       track           logs/log

Route  main      supplant        1
Route  main      receipt         etc/receipt.html
Route  main      master          log mail user
Route  main      cascade         log mail user
Route  main      encrypt_program  "/usr/bin/gpg -e -a r '%s' --batch"
```

This will have the following behavior:

Order

The master order route is *main*, the last one defined. It cascades the routes *log*, *mail*, and *user*, which means they will run in that order at the completion of the *main* route. The individual item routes `HARD` and `SOFT`, if applicable, will run before those.

Transactions

The route *log* specifies the tables that will be put in transaction mode, in this case `transactions orderline`, and `inventory`.

Failure

All order routes must succeed except *user*, which has `error_ok` set to 1.

Encryption The *mail* order route and the `HARD` order route will be sent by email, and encrypted against different GPG key IDs. They will get their `encrypt_program` setting from the main route.

To set the order routing for individual items, some method of determining their status must be made and the `mv_order_route` attribute must be set. This could be set at the time of the item being placed in the basket, or have a database field called `goods_type` set to the appropriate value. The following example uses a Perl routine on the final order form:

```
[perl table=products]
   my %route;
   my $item;
   foreach $item (@{$Items}) {
       my $code = $item->{code};
       my $keycode = $Tag->data('products', 'goods_type', $code);
       $item->{mv_order_route} = $keycode;
   }
   return;
[/perl]
```

Now the individual items are labeled with a `mv_order_route` value which causes their inclusion in the appropriate order routing.

Upon submission of the order form, any item labeled `HARD` will be accumulated and sent to the e-mail address `hardgoods@akopia.com`, where the item will be pulled from inventory and shipped.

Any item labeled `SOFT` will be passed to the order profile `create_download_link`, which will place it in a staging area for customer download. (This would be supported by a link on the receipt, possibly by reading a value set in the profile).

53. SSL Support

Interchange has several features that enable secure ordering via SSL (Secure Sockets Layer). Despite their mystique, SSL servers are actually quite easy to operate. The difference between the standard HTTP server and the SSL HTTPS server, from the standpoint of the user, is only in the encryption and the specification of the URL; `https:` is used for the URL protocol specification instead of the usual http: designation.

IMPORTANT NOTE: Interchange attempts to perform operations securely, but no guarantees or warranties of any kind are made! Since Interchange comes with Perl source, it is possible to modify the program to create security problems. One way to minimize this possibility is to record digital signatures, using MD5 or PGP, of `interchange`, `interchange.cfg`, and all modules included in Interchange. Check them on a regular basis to ensure they have not been changed.

Interchange uses the `SecureURL` directive to set the base URL for secure transactions, and the `VendURL` directive for normal non−secure transactions. Secure URLs can be enabled for forms through a form action of `[process−target secure=1]`. An individual page can be displayed via SSL with `[page href=mvstyle_pagename secure=1]`. A certain page can be set to be always secure with the `AlwaysSecure` catalog.cfg directive.

Interchange incorporates additional security for credit card numbers. The field `mv_credit_card_number` will not ever be written to disk.

To enable automated encryption of the credit card information, the directive `CreditCardAuto` needs to be defined as `Yes`. `EncryptProgram` also needs to be defined with some value, one which will, hopefully, encrypt the number. PGP is now recommended above all other encryption program. The entries should look something like:

```
CreditCardAuto   Yes
EncryptProgram   /usr/bin/pgpe −fat −r sales@company.com
```

See `CreditCardAuto` for more information on how to set the form variables.

54. Frequently Asked Questions

54.1. I can't get SQL to work: Undefined subroutine &Vend::Table::DBI::create ...

This probably means one of the following:

No SQL database.

Interchange doesn't include a SQL database. You must select one and install it.

No DBI.

You must install Perl's DBI module before using Interchange with SQL. You can see where to get it at `http://www.cpan.org`, or try:

```
perl -MCPAN -e 'install DBI'
```

No DBD.

You must install the specific Perl DBD module for your database before using Interchange with SQL. You can see where to get it at `http://www.cpan.org`, or try:

```
perl -MCPAN -e 'install DBD::XXXXX'
```

where XXXX is the name of your module. Some of them are:

```
Adabas
DB2
Informix
Ingres
ODBC
Oracle
Pg
Solid
Sybase
Unify
XBase
mSQL
mysql
```

If you can't make this script run without error:

```
use DBI;
use DBD::XXXXX;
```

Then you don't have one of the above, and Interchange can't use an SQL database until you get one installed.

I don't like the column types that Interchange defines!

They can be changed. See the `foundation/dbconf/mysql` directory for some examples under MySQL.

I change the ASCII file, but the table is not updated. Why?

Interchange writes an empty file `TABLE.sql` (where TABLE is the name of the table). When this is present, Interchange will never update the table from disk.
Also, if you have changed the field names in the file, you must restart the catalog (Apply Changes) before they will be picked up.

Why do I even need an ASCII file?

Interchange wants some source for column names initially. If you don't want to have one, just create a `TABLENAME.sql` file in the `products` directory. For example, if you have this:

```
Database products products.txt dbi:mysql:test_minivend
```

Then create a file `products/products.sql`.
\For:

```
Database pricing pricing.txt dbi:mysql:test_minivend
```

Create a file `products/pricing.sql`. .

Interchange overwrites my predefined table!

Yes, it will if you don't create a file called `TABLENAME.sql`, where `TABLENAME` is the name of the Interchange table. If you want this to happen by default, then set `NoImport TABLENAME`.

54.2. How can I use Interchange with Microsoft Access?

Though Interchange has ODBC capability, the Microsoft Access ODBC driver is not a network driver. You cannot access it on a PC from your ISP or UNIX system.

However, you can turn it around. Once you have created a MySQL or other SQL database on the UNIX machine, you may then obtain the Windows ODBC driver for the database (MySQL has a package called myODBC) and use the UNIX database as a data source for your PC−based database program.

Here is a quick procedure that might get you started:

- Get MySQL from:

  ```
  http://www.mysql.com/
  ```

  Install it on your UNIX box. On LINUX, it is as easy as getting the RPM distribution:

  ```
  http://www.mysql.com/rpm/
  ```

  You install it by typing, as root, `rpm −i mysql−3.XX.XX.rpm`. If you are not root, you will have to build the source distribution.
- To avoid permissions problems for your testing, stop the MySQL daemon and allow global read−write access with:

  ```
  mysqladmin shutdown
  ```

```
safe_mysqld --skip-grant-tables &
```

Obviously, you will want to study MySQL permissions and set up some security pretty quickly. It has excellent capability in that area, and the FAQ will help you get over the hurdles.

- Set up a database for testing on the UNIX machine:

```
mysqladmin create test_odbc
mysql test_odbc
```

Make an SQL query to set up a table, for example:

```
mysql> create table test_me ( code char(20), testdata char(20) );
Query OK, 0 rows affected (0.29 sec)

mysql> insert into test_me VALUES ('key1', 'data1');
Query OK, 1 rows affected (0.00 sec)

mysql> insert into test_me VALUES ('key2', 'data2');
Query OK, 1 rows affected (0.00 sec)

mysql>
```

- Get and install myODBC, also from the MySQL site:

```
http://www.mysql.com/
```

You install this package on your Windows 95 or NT box. It is a simple setup.exe process which leads you to the control panel for setting up an ODBC data source. Set up a data source named `test_odbc` that points to the database `test_odbc` on the UNIX box. You will need to know the host name and the port (usually 3306).

- With Microsoft Access, you can then open a blank database and select: File/Get External Data/Link Tables. Select File Type of 'ODBC databases' and the proper data source, and you should have access to the database residing on the UNIX side. line:

Interchange Tags Reference

55. Interchange Tag Reference

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

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

This document covers Interchange versions 4.7−4.8.

55.1. Tag Syntax

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

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

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

```
[value name]
```

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

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

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

55.1.1. Standard Syntax

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

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

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

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

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

55.1.1.1. HTML−Comment

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

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

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

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

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

```
'<!−−[value name]−−>'  becomes  'Bill'
'<!−− [value name] −−>'  becomes  '<!−− Bill −−>'
```

55.1.1.1.1. Technical notes

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

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

55.1.2. HTML−Embedded

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

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

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

HTML syntax:

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

Standard syntax:

```
[if items]
<TABLE>
```

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

Note -- Disabling HTML-embedded tags for performance:

Avoid the HTML-embedded usage if you can.

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

55.1.3. Named vs. Positional Parameters

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

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

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

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

```
[value foo]
```

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

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

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

55.1.3.1. Interpolating parameters

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

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

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

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

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

55.1.4. Universal Attributes

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

55.1.4.1. interpolate

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

For container tags (interpolated)

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

For standalone tags (reparsed)

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

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

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

Interpolation example:

Assuming that `name` is 'Kilroy',

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

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

Reparsing example:

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

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

If today is Monday, January 1, 2001,

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

the first line yields

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

while the second yields

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

55.1.4.2. reparse

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

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

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

Example:

Assuming that `name` is 'Kilroy',

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

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

expands to

```
1.  [value name] was here

2.  Kilroy was here
```

55.1.4.3. send

Deprecated

55.1.5. Tag−specific Attributes

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

55.1.6. Attribute Arrays and Hashes

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

```
attribute.n=value

attribute.hashkey=value
```

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

Here is an example of an attribute array:

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

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

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

55.1.6.1. Perl calls

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

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

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

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

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

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

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

55.2. Looping tags and Sub−tags

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

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

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

Tag	*Prefix*	*Examples*
[loop]	loop	[loop-code], [loop-field price], [loop-increment]
[item-list]	item	[item-code], [item-field price], [item-increment]
[search-list]	item	[item-code], [item-field price], [item-increment]
[query]	sql	[sql-code], [sql-field price], [sql-increment]

Sub-tag behavior is consistent among the looping tags.

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

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

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

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

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

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

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

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

]
```

You can also return hash lists in queries:

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

Now the data structure will look like:

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

55.2.1. Parse Order

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

55.2.2. [PREFIX−accessories arglist]

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

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

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

If the list is a hash list, i.e. an [item−list], then the value of the current item hash is passed so that a value default can be established.

55.2.3. [PREFIX−alternate N] DIVISIBLE [else] NOT DIVISIBLE [/else][/PREFIX−alternate]

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

For example:

```
[item−alternate 2]EVEN[else]ODD[/else][/item−alternate]
[item−alternate 3]BY 3[else]NOT by 3[/else][/item−alternate]
```

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

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

55.2.5. [PREFIX−change][condition] ... [/condition] TEXT [/PREFIX−change]

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

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

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

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

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

```
<TABLE>
```

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

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

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

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

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

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

55.2.6. [PREFIX−code]

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

55.2.7. [PREFIX−data table field]

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

55.2.8. [PREFIX−description]

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

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

55.2.9. [PREFIX−discount]

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

along with quantity pricing.

55.2.10. [PREFIX−discount_subtotal]

Inserts the discounted subtotal of the ordered items.

55.2.11. [PREFIX−field]

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

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

A common user error is to do this:

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

[loop−field foo_field]
[/loop]
```

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

55.2.12. [PREFIX−increment]

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

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

55.2.13. [PREFIX−last] CONDITION [/PREFIX−last]

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

55.2.14. [PREFIX−modifier attribute]

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

55.2.15. [PREFIX−next] CONDITION [/PREFIX−next]

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

55.2.16. [PREFIX−param name]

55.2.17. [PREFIX−pos N]

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

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

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

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

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

55.2.18. [PREFIX−price]

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

55.2.19. [PREFIX−quantity]

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

55.2.20. [PREFIX−subtotal]

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

55.2.21. [if−PREFIX−data table field] IF text [else] ELSE text [/else] [/if−PREFIX−data]

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

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

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

Careful, a space is not a false value!

55.2.22. [if−PREFIX−field field] IF text [else] ELSE text [/else] [/if−PREFIX−field]

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

55.2.23. [modifier−name attribute]

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

1. attribute0
2. attribute1
3. attribute2

etc.

55.2.24. [quantity−name]

Outputs for successive items in the list:

1. quantity0
2. quantity1
3. quantity2

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

56. Tags

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

56.1. accessories

A Swiss−army−knife widget builder, this provides access to Interchange's product option attributes (e.g., to choose or access product options such as a shirt's size or color).

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

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

56.1.1. Summary

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

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

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

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

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

Tag expansion example:

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

ASP−like Perl call:

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

or similarly with positional parameters,

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

56.1.1.1. See Also

Looping tags and Sub−tags.

56.1.2. Description

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

Interchange allows you to choose item attribute values for each ordered item −− you can attach a size, color, or other modifier to a line item in the shopping cart. You can also resubmit previous attribute values via hidden fields on a form.

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

```
UseModifier        size color
```

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

Important Note −− You may not use the following names for attributes:

item group quantity code mv_ib mv_mi mv_si

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

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

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

Interchange will automatically generate the select boxes when the [`accessories` code=os28044 attribute=size] or `[item−accessories size]` tags are called. They have the syntax:

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

[accessories code=sku
             attribute=modifier
             type=select
             column=db_table_column_name
             table=db_table
             name=varname
             outboard=key
             passed="value=label, value2*, value3=label 3" ]
```

```
[accessories js=| onChange="set_description(simple_options, variant)"; |
              type=select
              name="[item-param o_group]"
              passed="=--choose--,[item-param o_value]" ]
```

Notes:

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

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

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

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

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

```
[item-accessories color]
```

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

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

In combination with the `mv_order_item` and `mv_order_quantity` session variables, you can use this to allow a customer to enter an item attribute during an order.

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

The value can then be displayed with [`item-modifier` color] on the order report, order receipt, or any other page containing an [`item-list`].

56.1.2.1. Emulating with a loop

You can also build widgets directly, without using the `accessories` tag. You may have to do so if you need more control of the content than the tag offers. Below is a fragment from a shopping basket display form which shows a selectable size with "sticky" setting and a price that changes based upon the modifier setting. (Note that this example would normally be contained within the [`item list`][`/item-list`] pair.)

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

```
<OPTION> [loop-code] -- [price code="[item-code]" size="[loop-code]"]
[/loop]
</SELECT>
```

The output of the above would be similar to the output of [item-accessories size, select] if the product database field `size` contained the value `S, M, L, XL`. The difference is that the options in the loop emulation show the adjusted price in addition to the size within each option value.

56.1.2.2. Hash Lists -- Technical Note

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

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

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

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

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

56.1.2.3. code

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

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

56.1.2.4. arg

Deprecated after Interchange 4.6

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

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

Whitespace within the list is optional.

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

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

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

previous example might actually be something like the following:

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

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

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

56.1.2.5. attribute

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

This selects the item attribute the tag will work with.

56.1.2.6. type

This determines the action to be taken. One of:

Action	***Description***
select	Builds a dropdown <SELECT> menu for the item attribute, with the default item attribute value SELECTED. The accessories tag builds a select widget by default if type is not set.
display	Shows the label text for *only the selected option* if called in Hash List context (e.g., within an [item−list]). Ignored otherwise (i.e., the tag will build the default <SELECT> menu).
show	Returns the list of possible attributes for the item (without labels or any HTML widget). For example, if sku os28044 is available in several sizes: [accessories os28044 size,show] −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−− Sm=10oz, Med=15oz*, Lg=20oz
options	This shows the attribute options as a newline delimimited list: [accessories os28044 size,options] −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−− Sm Med Lg
labels	This shows the attribute option labels: [accessories os28044 size,options] −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−− 10oz 15oz* 20oz
radio	Builds a radio box group for the item, with spaces separating the elements.

radio nbsp	Builds a radio box group for the item, with separating the elements.
radio break	Builds a radio box group for the item, with ' ' separating the radio button/label pairs from one another.
radio left n	Builds a radio box group for the item, inside a table, with the checkbox on the left side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns. You can also set FONT SIZE like this: `type="radio left n fontsizem"` where −9 <= *m* <= 9
radio right n	Builds a radio box group for the item, inside a table, with the checkbox on the right side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns. You can also set FONT SIZE like this: `type="radio right n fontsizem"` where −9 <= *m* <= 9
check	Builds a checkbox group for the item, with spaces separating the elements.
check nbsp	Builds a checkbox group for the item, with ' ' separating the checkbox/label pairs from one another.
check break	Builds a checkbox group for the item, with ' ' separating the checkbox/label pairs from one another.
check left *n*	Builds a checkbox group for the item, inside a table, with the checkbox on the left side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns. You can also set FONT SIZE like this: `type="check left n fontsizem"` where −9 <= *m* <= 9
check right n	Builds a checkbox group for the item, inside a table, with the checkbox on the right side. If "n" is present and is a digit from 2 to 9, it will align the options in that many columns. You can also set FONT SIZE like this: `type="check right n fontsizem"` where −9 <= *m* <= 9
textarea_XX_YY	A textarea with XX columns and YY rows. The textarea will contain the selected item attribute value if used in `Hash List` context (e.g., within an [`item-list`]). If you simply use '`type`=textarea', the size will default to 4 rows by 40 columns, unless you have set the `rows` or `cols` tag attributes.
text_YY	A text box with YY width in characters. The HTML tag's VALUE will be set to the selected item attribute value if used in `Hash List` context (e.g., within an [`item-list`]).

	If you simply use 'type=text', the width will default to 60, unless you have set the cols tag attribute.
combo	Special type, used with nullselect filter, for selecting from a list or inputting a new value
reverse_combo	Special type, used with last_non_null filter, for selecting from a list or inputting a new value -- differs from combo in order of presentation
move_combo	Special type, used with null_to_space or null_to_comma filter, for selecting multiple non-ordered values from a list or inputting into a textarea
links	Produces a series of links based on the option values. The base form value is passed via the form parameter, just like in an [area ...] or [page ...] tag, and the value is named with the passed NAME attribute.
value	Returns the selected value if called in Hash List context (e.g., within an [item-list]), or nothing otherwise.
hidden	Creates a hidden form field. The hidden field's VALUE will be set to the selected item attribute value if used in Hash List context (e.g., within an [item-list]).
password_YY	A password box with YY width in characters. The HTML tag's VALUE will be set to the selected item attribute value if used in Hash List context (e.g., within an [item-list]). If you simply use 'type=password', the width will default to 12, unless you have set the cols tag attribute.

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

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

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

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

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

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

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

  will both output:

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

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

56.1.2.7. column

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

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

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

56.1.2.8. table

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

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

56.1.2.9. name

This sets the name of the form variable to use if appropriate for the widget being built. Defaults to 'mv_order_**attribute**' −− i.e. if the attribute is **size**, the form variable will be named **mv_order_size**.

If the variable is set in the user session, the widget will "remember" its previous setting. In other words,

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

56.1.2.10. outboard

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

56.1.2.11. passed

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

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

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

This will generate:

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

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

56.1.2.12. delimiter

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

56.1.2.13. prepend

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

For example,

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

56.1.2.14. append

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

56.1.2.15. extra

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

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

56.1.2.16. js

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

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

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

56.1.2.17. rows

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

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

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

56.1.2.18. cols

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

See also '`rows`' above.

56.1.2.19. width

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

56.1.2.20. default

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

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

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

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

56.1.2.21. price

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

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

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

56.1.2.22. contains

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

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

56.1.2.23. joiner

Requires '[type]=links'.

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

56.1.2.24. href

Requires 'type=links'.

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

56.1.2.25. template

Requires 'type=links'.

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

56.1.2.26. form

Requires 'type=links'.

This sets the base value for the form in a `links` type. Default is `mv_action=return`, which will simply set the variable value in the link.

For example, to generate a series of links −− one per item attribute value passed −− that set the variable "color" to the corresponding `passed` value (blank, blue, or green), do this:

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

This will generate something like the following:

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

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

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

56.1.2.27. empty

Requires 'type=links'.

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

56.1.2.28. secure

Requires 'type=links'.

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

56.1.2.29. new

Requires 'type=combo' or 'reverse_combo'.

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

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

Or, with the default new value:

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

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

56.2. and

56.2.1. Summary

Parameters: **type term op compare**

THIS TAG HAS SPECIAL POSITIONAL PARAMETER HANDLING.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

Called Routine for positonal:

ASP-like Perl call:

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

Attribute aliases

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

56.2.2. Description

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

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

See [if ...].

56.3. area

Alias: **href**

Expands to the URL for an Interchange page or action, including the Interchange session ID and supplied arguments. This is very similar to the page tag −− these are equivalent:

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

56.3.1. Summary

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

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

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

Other_Charactreristics	

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

Tag expansion example:

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

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

ASP-like Perl call:

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

or similarly with positional parameters,

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

56.3.1.1. See Also

page

56.3.2. Description

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

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

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

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

The optional *arg* is used just as in the page tag.

56.3.2.1. form

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

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

See the description of the page tag for more detail.

56.3.2.2. search

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

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

These are identical:

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

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

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

See the description of the page tag for more detail.

56.3.2.3. Examples

Tag expansion example:

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

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

Positional call example:

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

Named call example:

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

HTML−embedded example (disabled if no_html_parse Pragma active):

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

56.4. assign

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

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

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

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

56.4.1. Summary

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

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

Other_Charactreristics	
Invalidates cache	*No*
Container tag	*No*

ASP−like Perl call:

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

56.4.1.1. See Also

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

56.4.2. Description

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

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

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

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

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

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

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

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

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

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

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

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

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

56.4.2.1. clear

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

56.4.2.2. shipping

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

56.4.2.3. handling

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

Important note

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

56.4.2.4. salestax

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

56.4.2.5. subtotal

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

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

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

56.5. attr_list

56.5.1. Summary

Parameters: **hash**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.5.2. Description

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

Example:

```
        [perl tables=products]
                my %opt = (
                        hashref => 1,
                        sql => 'select * from  products',
                );
                my $ary_of_hash = $Db{products}->query(\%opt);
                my $template = <<EOF;
{sku} - {description} - {price|Call for price}
        {image?}<IMG SRC="{image}">{/image?}
        {image:}No image available{/image:}
EOF
                foreach my $ref (@$ary_of_hash) {
                        $out .= $Tag->attr_list($template, $ref);
                }
                return $out;
        [/perl]
```

Tags according to the following rules:

56.5.2.1. {key}

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

56.5.2.2. {key|fallback string}

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

56.5.2.3. {key true string}

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

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

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

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

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

56.6. banner

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

56.6.1. Summary

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

Parameters	***Description***	***Default***
category		default

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

Other_Charactreristics	
Invalidates cache	*No*
Container tag	*No*

Tag expansion example:

```
[banner category=]
```

ASP-like Perl call:

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

or similarly with positional parameters,

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

56.6.1.1. See Also

Banner/Ad rotation

56.6.2. Description

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

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

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

56.6.2.1. category

Default: category="default"

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

56.6.2.2. table

Default: table="banner"

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

56.6.2.3. r_field

Default: r_field="rotate"

Unweighted ads only.

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

56.6.2.4. b_field

Default: b_field="banner"

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

56.6.2.5. separator

Default: separator=":"

Unweighted ads only.

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

56.6.2.6. delimiter

Default: delimiter="{or}"

Unweighted ads only.

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

56.6.2.7. weighted

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

56.6.2.8. once

Weighted ads only.

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

56.6.2.9. c_field

Default: c_field="category"

Weighted ads only.

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

56.6.2.10. w_field

Default: w_field="weight"

Weighted ads only.

This specifies the column containing the banner weight.

56.7. bounce

56.7.1. Summary

Parameters: **href if**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

56.7.2. Description

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

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

Example of bouncing to an Interchange parsed page:

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

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

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

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

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

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

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

56.8. calc

Calculates the value of the enclosed arithmetic expression.

56.8.1. Summary

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

No parameters

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

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

ASP−like Perl call:

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

56.8.2. Description

Calculates the value of the enclosed arithmetic expression.

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

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

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

will expand to:

```
4
```

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

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

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

56.9. cart

56.9.1. Summary

Parameters: **name**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

OR

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

56.9.2. Description

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

56.10. catch

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

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

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

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

See also [`try`].

56.10.1. Summary

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

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

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

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

Attributes	*Default*
interpolate	*Yes*
reparse	*Yes*

Other_Charactreristics	
Invalidates cache	*No*
Container tag	Yes
Has Subtags	[*Error message text*] body [*/Error message text*]

Tag expansion example

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

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

```
Perl Safe error
```

ASP-like Perl call:

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

or similarly with positional parameters,

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

56.10.1.1. See Also

try

56.10.2. Description

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

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

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

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

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

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

56.10.2.1. label

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

56.11. cgi

56.11.1. Summary

Parameters: **name**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.11.2. Description

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

For instance, if you access the following URL:

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

`bar` will be substituted for `[cgi foo]`.

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

56.12. checked

56.12.1. Summary

Parameters: **name value**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

OR

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

56.12.2. Description

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

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

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

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

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

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

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

56.13. control

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

56.13.1. Summary

Parameters: **name default**

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

Attributes

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

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

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

OR

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

56.13.2. Description

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

56.14. control_set

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

56.14.1. Summary

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

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

Parameters: **index**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.14.2. Description

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

56.15. counter

56.15.1. Summary

Parameters: **file**

Positional parameters in same order.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

 OR

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

Attribute aliases

```
            name ==> file
```

56.15.2. Description

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

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

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

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

Additional parameters:

56.15.2.1. decrement=1

Causes the counter to count down instead of up.

56.15.2.2. start=50

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

56.15.2.3. value=1

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

56.16. currency

56.16.1. Summary

Parameters: **convert noformat**

Positional parameters in same order.

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

Interpolates **container text** by default>.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

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

OR

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

56.16.2. Description

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

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

will display:

```
4.00
```

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

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

will cause the number 8.982,04 to be displayed.

56.17. data

56.17.1. Summary

Parameters: **table field key**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

```
OR

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

Attribute aliases

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

56.17.2. Description

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

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

If a DBM−based database is to be modified, it must be flagged writable on the page calling the write tag. Use [tag flag write]products[/tag] to mark the `products` database writable, for example. **This must be done before ANY access to that table.**

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

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

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

56.18. default

56.18.1. Summary

Parameters: **name default**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

OR

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

56.18.2. Description

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

56.19. description

56.19.1. Summary

Parameters: **code base**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

OR

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

56.19.2. Description

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

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

56.20. discount

56.20.1. Summary

Parameters: **code**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

OR

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

56.20.2. Description

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

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

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

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

To apply a straight 20% discount to all items:

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

or with named attributes:

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

To take 25% off of only item 00-342:

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

To subtract $5.00 from the customer's order:

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

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

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

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

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

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

```
[item-list]
```

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

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

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

56.21. dump

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

56.21.1. Summary

No parameters.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP-like Perl call:

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

OR

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

56.21.2. Description

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

56.22. ecml

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

56.22.1. Summary

Parameters: **name function**

- function (default = 'widget')

```
Place form boxes on page:

         [ecml name]
         [ecml address]

Magic database entry from country database:

        [ecml country]

Map values back to Interchange variables for saving in UserDB:

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

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.22.2. Description

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

It translates into ECML from the following Interchange variables:

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

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

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

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

56.23. either

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

56.23.1. Summary

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

No parameters.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.23.2. Description

NO Description

56.24. error

56.24.1. Summary

Parameters: **name**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.24.2. Description

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

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

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

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

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

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

The options are:

56.24.2.1. all=1

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

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

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

56.24.2.2. joiner=*char*

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

56.24.2.3. keep=1

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

56.24.2.4. show_error=1

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

56.24.2.5. show_label=1

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

```
First Name: blank
```

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

show_label was added in 4.7.0.

56.24.2.6. show_var=1

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

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

56.24.2.7. std_label

std_label=<label string for error message>

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

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

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

56.24.2.8. required=1

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

56.25. export

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

56.25.1. Summary

Parameters: **table**

Positional parameters in same order.

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

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

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

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

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

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

OR

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

Attribute aliases

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

56.25.2. Description

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

56.26. field

56.26.1. Summary

Parameters: **name code**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

OR

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

Attribute aliases

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

56.26.2. Description

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

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

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

56.27. file

56.27.1. Summary

Parameters: **name type**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

OR

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

56.27.2. Description

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

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

56.28. filter

56.28.1. Summary

Parameters: **op**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

OR

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

56.28.2. Description

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

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

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

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

will display as

```
    Grant Wood
```

Another way to do this would be:

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

Filters available include:

56.28.2.1. cgi

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

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

56.28.2.2. digits

Returns only digits.

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

```
                return $val;
            },
```

56.28.2.3. digits_dot

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

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

56.28.2.4. dos

Turns linefeeds into carriage−return / linefeed pairs.

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

56.28.2.5. entities

Changes < to `<`, " to `"`, etc.

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

56.28.2.6. gate

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

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

56.28.2.7. lc

Lowercases the text.

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

56.28.2.8. lookup

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

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

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

```
    'lookup' => sub {
                        my ($val, $tag, $table, $column) = @_;
                        return tag_data($table, $column, $val) || $val;
                },
```

56.28.2.9. mac

Changes newlines to carriage returns.

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

56.28.2.10. name

Transposes a LAST, First name pair.

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

56.28.2.11. namecase

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

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

56.28.2.12. no_white

Strips all whitespace.

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

56.28.2.13. pagefile

Strips leading slashes and dots.

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

```
            },
```

56.28.2.14. sql

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

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

56.28.2.15. strip

Strips leading and trailing whitespace.

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

56.28.2.16. text2html

Rudimentary HTMLizing of text.

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

56.28.2.17. uc

Uppercases the text.

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

56.28.2.18. unix

Removes those crufty carriage returns.

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

56.28.2.19. urlencode

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

```
'urlencode' => sub {
                my $val = shift;
                $val =~ s|[^\w:]|sprintf "%%%02x", ord $1|eg;
                return $val;
            },
```

56.28.2.20. value

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

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

56.28.2.21. word

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

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

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

```
package Vend::Interpolate;

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

That filter will reverse the characters sent.

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

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

56.29. flag

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

56.29.1. Summary

Parameters: **type**

`type` may be one of the following

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

56.29.2. Attributes

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

 OR

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

Attribute aliases

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

56.29.3. Description

The `flag` tag controls database access and transactions.

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

For example, you can call

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

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

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

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

56.30. fly_list

56.30.1. Summary

Parameters: **code base**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.30.2. Description

Syntax: [fly−list prefix=tag_prefix* code=code*]

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

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

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

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

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

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

56.31. fly_tax

56.31.1. Summary

Parameters: **area**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

OR

$Tag−>fly_tax($area);
```

56.31.2. Description

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

56.32. goto

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

56.32.1. Summary

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

or positionally,

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

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

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

ASP−like Perl call:

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

56.32.2. Description

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

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

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

56.32.2.1. name

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

56.32.2.2. if

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

For example, this will not execute the goto:

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

56.33. handling

56.33.1. Summary

Parameters: **mode**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

OR

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

Attribute aliases

```
            carts ==> cart
            modes ==> mode
```

```
name ==> mode
tables ==> table
```

56.33.2. Description

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

56.34. harness

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

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

56.34.1. Summary

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

No parameters.

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

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

 OR

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

56.34.2. Description

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

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

56.35. href

Alias for [area] tag.

56.36. html_table

Builds an HTML table

56.36.1. Summary

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

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

```
        {
        },
        BODY
    )

OR

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

56.36.2. Description

Builds an HTML table

56.37. if

56.37.1. Summary

Parameters: **type term op compare**

THIS TAG HAS SPECIAL POSITIONAL PARAMETER HANDLING.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **YES**

Called Routine:

Called Routine for positonal:

ASP−like Perl call:

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

Attribute aliases

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

56.37.2. Description

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

Positional call example: [if type field op compare]

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

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

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

```
[if type term op compare]
[then]
                            If true, this is printed on the document.
                            The [then] [/then] is optional in most
                            cases. If ! is prepended to the type
                            setting, the sense is reversed and
                            this will be output for a false condition.
[/then]
[elsif type term op compare]
                            Optional, tested when if fails
[/elsif]
[else]
                            Optional, printed when all above fail
[/else]
[/if]
```

The [if] tag can also have some variants:

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

You can do some Perl−style regular expressions:

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

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

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

Oops! This will not work. You must do instead

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

or better yet

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

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

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

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

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

There are many test targets available:

56.37.2.1. config Directive

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

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

56.37.2.2. data database::field::key

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

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

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

56.37.2.3. discount

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

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

56.37.2.4. explicit

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

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

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

This will work for *Variable* values:

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

56.37.2.5. file

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

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

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

56.37.2.6. items

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

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

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

56.37.2.7. ordered

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

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

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

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

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

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

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

56.37.2.8. pragma

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

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

56.37.2.9. scratch

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

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

56.37.2.10. session

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

56.37.2.11. validcc

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

56.37.2.12. value

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

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

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

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

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

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

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

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

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

56.37.2.13. [then] text [/then]

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

56.37.2.14. [elsif type field op* compare*]

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

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

56.37.2.15. [else] text [/else]

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

56.37.2.16. [condition] text [/condition]

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

arguments are in the [*per*l] construct.

56.38. import

56.38.1. Summary

Parameters: **table type**

Positional parameters in same order.

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

Interpolates **container text** by default>.

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

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

 OR

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

Attribute aliases

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

56.38.2. Description

Named attributes:

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

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

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

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

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

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

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

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

56.39. include

56.39.1. Summary

Parameters: **file locale**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP-like Perl call:

Not applicable.

56.39.2. Description

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

56.40. index

Creates an index for the specified table.

56.40.1. Summary

Parameters: **table**

Positional parameters in same order.

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

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

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

 OR

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

Attribute aliases

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

56.40.2. Description

Creates an index for the specified table.

56.41. item_list

56.41.1. Summary

Parameters: **name**

ONLY THE PARAMETERS ARE POSITIONAL.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

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

                 for(@$Items) { CODE }

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

 OR

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

Attribute aliases

```
            cart ==> name
```

56.41.2. Description

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

NOTE: The special tags that reference item within the list are not normal Interchange tags, do not take named attributes, and cannot be contained in an HTML tag (other than to substitute for one of its values or provide a

conditional container). They are interpreted only inside their corresponding list container. Normal Interchange tags can be interspersed, though they will be interpreted *after* all of the list−specific tags.

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

56.41.2.1. [if−data table column]

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

56.41.2.2. [if−data ! table column]

Reverses sense for [if−data].

56.41.2.3. [/if−data]

Terminates an [if_data table column] element.

56.41.2.4. [if−field fieldname]

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

56.41.2.5. [if−field ! fieldname]

Reverses sense for [if−field].

56.41.2.6. [/if−field]

Terminates an [if_field fieldname] element.

56.41.2.7. [item−accessories attribute*, type*, field*, database*, name*]

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

56.41.2.8. [item−code]

Evaluates to the product code for the current item.

56.41.2.9. [item−data database fieldname]

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

56.41.2.10. [item−description]

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

In support of `OnFly`, if the description field is not found in the database, the `description` setting in the shopping cart will be used instead.

56.41.2.11. [item−field fieldname]

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

56.41.2.12. [item−increment]

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

56.41.2.13. [item−last]tags[/item−last]

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

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

If this is contained in your [`item−list`] (or [`search−list`] or flypage) and the weight field is empty, then a numerical `−1` will be output from the [calc][/calc] tags; the list will end and the item will **not** be shown. If the product's weight field is less than 1, a numerical 1 is output. The item will be shown, but will be the last item shown. (If it is an [`item−list`], any price for the item will still be added to the subtotal.) NOTE: no HTML style.

56.41.2.14. [item−modifier attribute]

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

56.41.2.15. [item−next]tags[/item_next]

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

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

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

56.41.2.16. [item−price n* noformat*]

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

56.41.2.17. [discount−price n* noformat*]

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

56.41.2.18. [item−discount]

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

56.41.2.19. [item−discount_subtotal]

Inserts the discounted subtotal of the ordered items.

56.41.2.20. [item−quantity]

Evaluates to the quantity ordered for the current item.

56.41.2.21. [item−subtotal]

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

56.41.2.22. [modifier−name attribute]

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

56.41.2.23. [quantity−name]

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

56.42. label

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

Parameter: **name**

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

56.43. log

Log contained text to specified file.

56.43.1. Summary

Parameters: **file**

- file

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

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

OR

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

Attribute aliases

```
            arg ==> file
```

56.43.2. Description

Log contained text to specified file.

56.44. loop

56.44.1. Summary

Parameters: **list**

Positional parameters in same order.

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

Must pass named parameter interpolate=1 to cause interpolation.

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

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

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

                 for(@list) { CODE }

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

 OR

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

Attribute aliases

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

56.44.2. Description

HTML example:

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

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

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

```
    [loop list="A B C"]
```

```
    [loop prefix=mid list="[loop-code]1 [loop-code]2 [loop-code]3"]
        [loop prefix=inner list="X Y Z"]
            [mid-code]-[inner-code]
        [/loop]
    [/loop]
[/loop]
```

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

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

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

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

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

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

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

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

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

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

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

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

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

56.44.2.1. [if-loop-data table column] IF [else] ELSE [/else][/if-loop-field]

Outputs the IF if the `column` in `table` is non-empty, and the ELSE (if any) otherwise.

See [if-PREFIX-data].

56.44.2.2. [if-loop-field column] IF [else] ELSE [/else][/if-loop-field]

Outputs the **IF** if the `column` in the `products` table is non-empty, and the **ELSE** (if any) otherwise. Will fall through to the first non-empty field if there are multiple `ProductFiles`.

See [if-PREFIX-field].

56.44.2.3. [if-loop-param param] IF [else] ELSE [/else][/if-loop-param]

Only works if you have named return fields from a search (or from a passed list with the lr=1 parameter).

Outputs the **IF** if the returned `param` is non-empty, and the **ELSE** (if any) otherwise.

See [if-PREFIX-param].

56.44.2.4. [if-loop-pos N] IF [else] ELSE [/else][/if-loop-param]

Only works if you have multiple return fields from a search (or from a passed list with the lr=1 parameter).

Parameters are numbered from ordinal 0, with [loop-pos 0] being the equivalent of [loop-code].

Outputs the **IF** if the returned positional parmeter `N` is non-empty, and the **ELSE** (if any) otherwise.

See [if-PREFIX-pos].

56.44.2.5. [loop-accessories]

Outputs an [accessories ...] item.

See [PREFIX-accessories].

56.44.2.6. [loop-change marker]

See [PREFIX-change].

56.44.2.7. [loop-code]

Evaluates to the code for the current item.

See [PREFIX-code].

56.44.2.8. [loop−data database fieldname]

Evaluates to the field name *fieldname* in the arbitrary database table *database*, for the current item.

See [PREFIX−data].

56.44.2.9. [loop−description]

Evaluates to the product description (from the products file, passed description in on−fly item, or description attribute in cart) for the current item.

See [PREFIX−description].

56.44.2.10. [loop−field fieldname]

Evaluates to the field name *fieldname* in the database, for the current item.

See [PREFIX−field].

56.44.2.11. [loop−increment]

Evaluates to the number of the item in the list. Used for numbering items in the list.

Starts from integer 1.

See [PREFIX−increment].

56.44.2.12. [loop−last]tags[/loop−last]

Evaluates the output of the Interchange tags encased inside, and if it evaluates to a numerical non−zero number (i.e. 1, 23, or −1) then the loop iteration will terminate. If the evaluated number is **negative**, then the item itself will be skipped. If the evaluated number is **positive**, then the item itself will be shown but will be last on the list.

```
[loop−last][calc]
  return −1 if '[loop−field weight]' eq '';
  return 1 if '[loop−field weight]' < 1;
  return 0;
  [/calc][/loop−last]
```

If this is contained in your `[loop list]` and the weight field is empty, then a numerical `−1` will be output from the [calc][/calc] tags; the list will end and the item will **not** be shown. If the product's weight field is less than 1, a numerical 1 is output. The item will be shown, but will be the last item shown.

56.44.2.13. [loop−next]tags[/loop−next]

Evaluates the output of the Interchange tags encased inside, and if it evaluates to a numerical non−zero number (i.e. 1, 23, or −1) then the loop will be skipped with no output. Example:

```
[loop−next][calc][loop−field weight] < 1[/calc][/loop−next]
```

If this is contained in your `[loop list]` and the product's weight field is less than 1, then a numerical `1` will be output from the [calc][/calc] operation. The item will not be shown.

56.44.2.14. [loop−price n* noformat*]

Evaluates to the price for optional quantity n (from the products file) of the current item, with currency formatting. If the optional "noformat" is set, then currency formatting will not be applied.

56.45. mail

Mail contained text to recipient specified by 'to' using the program specified with the SendMailProgram catalog directive.

56.45.1. Summary

Parameters: **to**

Positional parameters in same order.

- raw
  - ♦ Send it raw without creating headers and checking content, recipient, subject, etc.
- extra
  - ♦ Additional headers (these will also be added to 'raw' messages)
- success
  - ♦ Tag return value if successful (default is 1).
- hide
  - ♦ Suppress tag return value. This would otherwise be the 'success' attribute setting.
- show
  - ♦ The tag will return the final message with headers in the page

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [mail] FOO [/mail]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->mail(
        {
         to => VALUE,
        },
        BODY
    )

 OR
```

```
$Tag->mail($to, $ATTRHASH, $BODY);
```

56.45.2. Description

Mail contained text to recipient specified by 'to' using the program specified with the SendMailProgram catalog directive.

56.46. mvasp

Executes the ASP−style perl code contained by the tag. The code will run under the restrictions of the Safe module. This is very similar to the [perl] tag, except that the standard '<%' and '%>' ASP delimiters allow you to mix HTML and perl code.

56.46.1. Summary

```
[mvasp tables] ASP here [/mvasp]
[mvasp tables="db1 db2 ..." other_named_attributes] ASP here [/mvasp]
```

Parameters	*Description*	*Default*
tables	Database tables to be made available to ASP Perl code	*none*
table	Alias for `tables`	*none*

Attributes	*Default*
failure	*none*
no_return	*Always true*
subs	*No*
arg="subs"	*Same as* `subs`
global	*No*
file	*none*
interpolate	*No*
reparse	*No*

Other_Charactreristics	
Invalidates cache	*Yes*
Has Subtags	<% *and* %>
Container tag	*Yes*
Nests	*No*

Tag expansion example:

```
[mvasp tables="products" failure="ASP Broke <BR>"]
   <P>This is HTML</p>
   <% my $sku = $Values->{code}; %>
   <P>More HTML</p>
   <% my $result = "Looked up SKU $sku. It is a ";
      $result .= $Tag->data('products', 'description', $sku );
      $Document->write( "$result <br>\n" ); %>
   <P>Still more HTML</p>
[/mvasp]
```

```
-----------------------------------------------------
      <P>This is HTML</p>

      <P>More HTML</p>
      Looked up SKU os28044. It is a Framing Hammer <br>

      <P>Still more HTML</p>
```

56.46.1.1. See Also

perl, Interchange Programming

56.46.2. Description

Executes the ASP−style perl code contained by the tag. The code will run under the restrictions of the Safe module. This is very similar to the [perl no_return=1] tag, except that the standard '<%' and '%>' ASP delimiters allow you to mix HTML and perl code.

See the perl tag and ASP−Like Perl sections for more detail.

56.46.2.1. tables

Whitespace−separated list of database tables to make available within the ASP−Perl code. See perl tag.

56.46.2.2. failure

The value the tag should return in case the perl code fails the eval. See perl tag.

56.46.2.3. no_return

The return value of the perl code is always suppressed. If you want output from the ASP code sections, you must explicitly write it with the &HTML or $Document−>write() functions.

You can also retrieve the return value of the perl code from the session hash via `[data session mv_perl_result]`. See perl tag.

56.46.2.4. subs

Enable GlobalSub routines (requires catalog directive AllowGlobal). See perl tag.

56.46.2.5. global

Turn off Safe protection (requires catalog directive AllowGlobal). See perl tag.

56.46.2.6. file

Prepend the contents of the specified file or FileDatabase entry to the perl code before `eval`'ing it. See perl tag.

56.46.2.7. Examples

See the ASP−Like Perl section of Interchange Programming.

56.47. nitems

Returns the total number of items ordered. Uses the current cart if none specified.

56.47.1. Summary

Parameters: **name**

- name
  - Cart name
  - Default: current cart
- qualifier
  - An item attribute that must be true in order to count the item.
  - Default: None
- compare
  - Regular expression the specified qualifier attribute's value must match to be counted. This replaces the truth value comparison.
  - Default: None (uses truth value of the specified qualifier attribute)

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->nitems(
        {
         name => VALUE,
        }
    )

 OR

    $Tag->nitems($name, $ATTRHASH);
```

56.47.2. Description

Expands into the total number of items ordered so far. Takes an optional cart name as a parameter.

56.48. options

Builds HTML widgets as defined in the options table for selecting options associated with a given product. This tag handles simple, matrix or modular options. See also the `accessories` tag.

Here is an illustrative example from the 'tools' sample data set of the foundation catalog:

```
===
   [options code=os28005]
---
   <input type=hidden name=mv_item_option value="logo">
     <SELECT NAME="mv_order_logo">
     <OPTION VALUE="c">Construct Something
     <OPTION VALUE="y" SELECTED>Your Logo</SELECT><BR>
   <input type=hidden name=mv_item_option value="color">
     <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BLK" > Black
     <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BEIGE" > Beige
     <INPUT TYPE="radio" NAME="mv_order_color" VALUE="WHITE" > White<BR>
   <input type=hidden name=mv_item_option value="bristle">
     <SELECT NAME="mv_order_bristle">
     <OPTION VALUE="synthetic">Synthetic
     <OPTION VALUE="camel">Camel Hair</SELECT>
===
```

56.48.1. Summary

Parameters: **code**

- code
  - Product key (usually sku).
  - No default
- table
  - Table to use for option attributes.
  - Default: 'options'
- td
  - Results as table rows. For example, compare the following example from the 'tools' sample data set with the earlier example:

```
===
   [options code=os28005 td=1]
---
   <td><input type=hidden name=mv_item_option value="logo">
      <SELECT NAME="mv_order_logo">
      <OPTION VALUE="c">Construct Something
      <OPTION VALUE="y" SELECTED>Your Logo</SELECT></td>
   <td><input type=hidden name=mv_item_option value="color">
      <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BLK" > Black
      <INPUT TYPE="radio" NAME="mv_order_color" VALUE="BEIGE" > Beige
      <INPUT TYPE="radio" NAME="mv_order_color" VALUE="WHITE" > White</td>
   <td><input type=hidden name=mv_item_option value="bristle">
      <SELECT NAME="mv_order_bristle">
      <OPTION VALUE="synthetic">Synthetic
      <OPTION VALUE="camel">Camel Hair</SELECT></td>
===
```

(Note that the output was reformatted to fit this page)

- price
  - Default: False
  - Boolean. If set and the options have prices, the HTML widget(s) will show the prices. This is like the `price` attribute of the accessories tag.
    Note that the `price_data` setting comes from the 'price' column of the options table.
    Technical note−− If your options table has different mappings, you can control this with

$::Variable->{MV_OPTION_TABLE_MAP}

- label
  - Shows labels for the options with the widgets.
  - Default: False

The following example (using another item from the 'tools' data) illustrates the price and label attributes:

```
===
   [options code=os28011 label=1 price=1]
---
   Handle<BR>
     <input type=hidden name=mv_item_option value="handle">
       <SELECT NAME="mv_order_handle">
       <OPTION VALUE="W">Wood handle
       <OPTION VALUE="E">Ebony handle ($20.00)</SELECT><BR>
   Blade material<BR>
     <input type=hidden name=mv_item_option value="blade">
       <SELECT NAME="mv_order_blade">
       <OPTION VALUE="P">Plastic blade ($-1.22)
       <OPTION VALUE="S" SELECTED>Steel blade
       <OPTION VALUE="T">Titanium blade ($100.00)</SELECT>
===
```

(again, the output has been reformatted to fit the page).

- bold
  - Boldfaces the labels if the `'label'` option is set.
  - Default: False
- joiner
  - The joiner for the widgets.
  - Default:

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP-like Perl call:

```
    $Tag->options(
        {
         code => VALUE,
        }
    )

 OR

    $Tag->options($code, $ATTRHASH);
```

56.49. or

56.49.1. Summary

Parameters: **type term op compare**

THIS TAG HAS SPECIAL POSITIONAL PARAMETER HANDLING.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

Called Routine for positonal:

ASP−like Perl call:

```
    $Tag−>or(
        {
         type => VALUE,
         term => VALUE,
         op => VALUE,
         compare => VALUE,
        }
    )

 OR

    $Tag−>or($type, $term, $op, $compare);
```

Attribute aliases

```
            base ==> type
            comp ==> compare
            operator ==> op
```

56.49.2. Description

NO Description

56.50. order

Expands into a hypertext link which will include the specified item in the list of products to order and display the order page.

56.50.1. Summary

```
    [order code quantity]Link Text[/order]
    [order code=os28044 quantity=2]Link Text</A>
```

Parameters	*Description*	*Default*

code	This is the unique identifier for the item, typically the SKU in the products table	*none*
quantity	Quantity to order	1

Attributes	***Default***
interpolate (reparse)	*No*

Other_Charactreristics	
Invalidates cache	*No*
Container tag	*No*
Has end tag	*No* (`[/order]` is a macro for `</A>`)

Tag expansion example:

```
   [order os28044 2]Buy Framing Hammer[/order]
---
   <A HREF="http://localhost.localdomain/cgi-bin/tag72/ord/basket?\
   mv_session_id=6CZ2whqo&mv_pc=1&mv_action=refresh&\
   mv_order_item=os28044&mv_order_quantity=3">Buy Framing Hammer</A>
```

ASP-like Perl call:

```
    $Tag->order($code, $quantity);
```

56.50.2. Description

Expands into a hypertext link which will include the specified code in the list of products to order and display the order page. **code** should be a product code listed in one of the "products" databases.

56.50.3. How to Order an Item

Interchange can either use a form-based order or a link-based order to place an item in the shopping cart. The `order` tag creates a link-based order.

You can use the `area` tag with form variables if you need more control, for example, to change frames for the order:

```
   <A HREF="[area href=ord/basket
                  form="mv_order_item=os28044
                        mv_order_quantity=2
                        mv_action=refresh"]"
    TARGET=newframe> Order Framing Hammer</A>
```

To order with a form, you set the form variable `mv_order_item` to the item-code/SKU and use the `refresh` action:

```
   <FORM ACTION="[process-target]" METHOD=POST>
   <INPUT TYPE=hidden  NAME="mv_todo"        VALUE="refresh">
   <INPUT TYPE=hidden  NAME="mv_order_item"  VALUE="os28044">

   Order <INPUT NAME="mv_order_quantity" SIZE=3 VALUE=1> Framing Hammer

   <INPUT TYPE=submit VALUE="Order!">
   </FORM>
```

Groups of items may be batched:

```
<FORM ACTION="[process-target]" METHOD=POST>
<INPUT TYPE=hidden  NAME="mv_todo"        VALUE="refresh">

<INPUT TYPE=hidden  NAME="mv_order_item"  VALUE="TK112">
<INPUT NAME="mv_order_quantity" SIZE=3> Standard Toaster

<INPUT TYPE=hidden  NAME="mv_order_item"  VALUE="TK200">
<INPUT NAME="mv_order_quantity" SIZE=3> Super Toaster

<INPUT TYPE=submit VALUE="Order!">
</FORM>
```

Items that have a quantity of zero (or blank) will be skipped. Only items with a positive quantity will be placed in the basket.

Attributes like size or color may be specified at time of order. See the `accessories` tag for detail.

56.51. page

Expands to a hyperlink to an Interchange page or action, including surrounding <A HREF ...>. The URL within the link includes the Interchange session ID and supplied arguments. The optional [/page] is simply a macro for </A>.

If you do not want the <A HREF ...>, use the area tag instead −− these are equivalent:

```
[page href=dir/page arg=mv_arg]TargetName[/page]
<A HREF="[area href=dir/page arg=mv_arg]">TargetName</A>
```

56.51.1. Summary

```
[page href arg]
[page href=dir/page arg=page_arguments other_named_attributes]
```

Parameters	*Description*	*Default*
href	Path to Interchange page or action *Special arguments* ♦ 'scan' treats **arg** as a search argument ♦ 'http://...' external link (requires **form** attribute)	process
arg	Interchange arguments to page or action	*none*
base	alias for **arg**	*none*

Attributes	*Default*
extra	*none*
form	*none*
search	*No*

secure	*No*
interpolate (reparse)	*No*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	*No* ([/page] is a macro for </A>)

Tag expansion example:

```
[page href=dir/page.html arg="arg1=AA/arg2=BB"]

<a href="www.here.com/cgi-bin/mycatalog/page.html?mv_session_id=6CZ2whqo&\
mv_pc=1&mv_arg=arg1%3dAA/arg2%3dBB">
```

ASP-like Perl call:

```
$Tag->page(  { href => "dir/page",
               arg  => "arguments", }  );
```

or similarly with positional parameters,

```
$Tag->page($href, $arg, $attribute_hash_reference);
```

Using arrayref for joined search (see also Attribute Arrays and Hashes)

```
my $searchref = [ "se=hammer/fi=products/sf=description",
                  "se=plutonium/fi=products/sf=description", ];

$Tag->page( { href   => 'scan',
              search => $searchref, } );
```

56.51.1.1. See Also

area

56.51.2. Description

The page tag inserts a hyperlink to the specified Interchange page or action. For example, [page shirts] will expand into

```
<a href="http://www.here.com/cgi-bin/mycatalog/shirts?mv_session_id=6CZ2whqo&mv_pc=1">
```

The catalog page displayed will come from "shirts.html" in the pages directory.

The additional argument will be passed to Interchange and placed in the {arg} session parameter. This allows programming of a conditional page display based on where the link came from. The argument is then available with the tag [data session arg], or the embedded Perl session variable $Session->{arg}. Spaces and some other characters will be escaped with the %NN HTTP-style notation and unescaped when the argument is read back into the session.

For better performance, Interchange can prebuild and cache pages that would otherwise be generated dynamically. If Interchange has built such a static page for the target, the page tag produces a link to the

cached page whenever the user has accepted and sent back a cookie with the session ID. If the user did not accept the cookie, Interchange cannot use the cache, since the link must then include the *mv_session_id* argument in order to preserve session.

56.51.2.1. form

The optional `form` argument allows you to encode a form in the link.

```
[page form="mv_order_item=os28044
            mv_order_size=15oz
            mv_order_quantity=1
            mv_separate_items=1
            mv_todo=refresh"] Order 15oz Framing Hammer</A>
```

The two form values *mv_session_id* and *mv_arg* are automatically added when appropriate. The form value *mv_arg* receives the value of the tag's `arg` parameter.

This would generate a form that ordered quantity one of item number `os28044` with size `15oz`. The item would appear on a separate line in the shopping cart, since `mv_separate_items` is set. Since the `href` is not set, you will go to the default shopping cart page -- alternatively, you could have set `mv_orderpage=yourpage` to go to `yourpage`.

All normal Interchange form caveats apply -- you must have an action, you must supply a page if you don't want to go to the default, etc.

You can theoretically submit any form with this, though none of the included values can have newlines or trailing whitespace. If you want to do something like that you will have to write a UserTag.

If the parameter `href` is not supplied, *process* is used, causing normal Interchange form processing.

If the `href` points to an http:// link, then no Interchange URL processing will be done, but the URL will include mv_session_id, mv_pc, and any arguments supplied with the `arg` attribute:

```
[page  href="http://www.elsewhere.net/cgi/script"
       form="cgi_1=ONE
             cgi_2=TWO"
        arg="Interchange argument"]External link</A>

<A HREF="http://www.elsewhere.net/cgi/script?\
 mv_session_id=6CZ2whqo&mv_pc=1&mv_arg=Interchange%20argument&\
 cgi_1=ONE&cgi_2=TWO">External link</A>
```

56.51.2.2. search

Interchange allows you to pass a search in a URL. There are two ways to do this:

1. Place the search specification in the named `search` attribute.
   - Interchange will ignore the `href` parameter (the link will be set to 'scan'.
   - If you give the `arg` parameter a value, that value will be available as [value mv_arg] within the search display page.
2. Set the `href` parameter to 'scan' and set `arg` to the search specification.
   - Note that you can use this form positionally -- the values go into `href` and `arg`, so you do not have to name parameters.

These are identical:

```
[page scan
      se=Impressionists
      sf=category]
   Impressionist Paintings
[/page]

[page href=scan
       arg="se=Impressionists
            sf=category"]
   Impressionist Paintings
</A>

[page search="se=Impressionists
              sf=category"]
   Impressionist Paintings
[/page]
```

Here is the same thing from a non−Interchange page (e.g., a home page), assuming '/cgi−bin/mycatalog' is the CGI path to Interchange's vlink):

```
<A HREF="/cgi−bin/mycatalog/scan/se=Impressionists/sf=category">
   Impressionist Paintings
</A>
```

Sometimes, you will find that you need to pass characters that will not be interpreted positionally. In that case, you should quote the arguments:

```
[page href=scan
      arg=|
            se="Something with spaces"
      |]
```

See the Search and Form Variables appendix for a listing of the form variables along with two−letter abbreviations and descriptions.

They can be treated just the same as form variables on the page, except that they can't contain spaces, '/' in a file name, or quote marks. These characters can be used in URL hex encoding, i.e. %20 is a space, %2F is a `/`, etc. −− `&sp;` or ` ` will not be recognized. If you use one of the methods below to escape these "unsafe" characters, you won't have to worry about this.

You may specify a one−click search in three different ways. The first is as used in previous versions, with the scan URL being specified completely as the page name. The second two use the "argument" parameter to the `[page ...]` or [`area` ...]> tags to specify the search (an argument to a scan is never valid anyway).

56.51.2.3. Original syntax

If you wish to do an OR search on the fields category and artist for the strings "Surreal" and "Gogh", while matching substrings, you would do:

```
[page scan se=Surreal/se=Gogh/os=yes/su=yes/sf=artist/sf=category]
   Van Gogh −− compare to surrealists
[/page]
```

In this method of specification, to replace a / (slash) in a file name (for the sp, bd, or fi parameter) you must use the shorthand of ::, i.e. sp=results::standard. (This may not work for some browsers, so you should probably either put the page in the main pages directory or define the page in a search profile.)

56.51.2.4. Ampersand syntax

You can substitute & for / in the specification and be able to use / and quotes and spaces in the specification.

```
[page scan se="Van Gogh"&sp=lists/surreal&os=yes&su=yes&sf=artist&sf=category]
   Van Gogh -- compare to surrealists
[/page]
```

Any "unsafe" characters will be escaped.

56.51.2.5. Multi−line syntax

You can specify parameters one to a line, as well.

```
[page scan
    se="Van Gogh"
    sp=lists/surreal
    os=yes
    su=yes
    sf=artist
    sf=category
] Van Gogh -- compare to surrealists [/page]
```

Any "unsafe" characters will be escaped. You may not search for trailing spaces in this method; it is allowed in the other notations.

56.51.2.6. Joined searches

You can also specify a joined search using an attribute array (see Attribute Arrays and Hashes):

```
[page href=scan
  search.0="se=fragrant
            fi=products
            sf=smell"
  search.1="se=purple
            sf=color"
  search.2="se=perennial
            sf=type"]
```

The search routine called by the page tag automatically adds the other relevant search specification elements, including the 'co=yes' to indicate a combined search (joined searches are described in the Interchange database documentation).

56.51.2.7. [/page]

This is not an actual end tag, but simply a macro that expands to </A>. The following two lines are equivalent:

```
[page shirts]Our shirt collection[/page]
[page shirts]Our shirt collection</A>
```

Tip: In large pages, just use the </A> tag for a small performance improvement.

56.52. perl

Executes the perl code contained by the tag. The code will run under the restrictions of Perl's Safe module by default. The tag expands to the value returned by the enclosed code (i.e., printing to STDOUT or STDERR is useless).

See also Interchange Programming.

56.52.1. Summary

```
[perl tables] Code here [/perl]
[perl tables="db1 db2 ..." other_named_attributes] Code here [/perl]
```

Parameters	*Description*	*Default*
tables	Database tables to be made available to ASP Perl code	*none*
table	Alias for `tables`	*none*

Attributes	*Default*
failure	*none*
no_return	*No*
subs	*No*
arg="subs"	*Same as* `subs`
global	*No*
file	*none*
number_errors	*none*
eval_label	*none*
short_errors	*none*
trim_errors	*none*
interpolate	*No*
reparse	*Yes*

Other_Charactreristics	
Invalidates cache	*Yes*
Has Subtags	*No*
Container tag	*Yes*

Tag expansion example:

```
   [perl tables="products" failure="Perl code error <BR>"]
      my $result = "Looked up SKU $Values->{code}. It is a ";
      $result .= $Tag->data('products', 'description', $Values->{code} );
      return ("$result <br>\n");
   [/perl]
-------------------------------------------------------
   Looked up SKU os28044. It is a Framing Hammer <br>
```

ASP−like Perl call: (e.g., to use it like a runtime eval() within your code)

```
$Tag->perl(  { tables  => "products", },
             $code  );
```

or similarly with positional parameters,

```
$Tag->perl( $tables, $attribute_hash_reference );
```

56.52.1.1. See Also

See also Interchange Programming, [calc], and [mvasp].

56.52.2. Description

This tag allows you to embed perl code within an Interchange page. The code will run under the restrictions of Perl's Safe module by default. Perl's '`warnings`' and '`strict`' pragmas are both turned off, and Safe will block you from turning them on, since it blocks Perl's '`use`' command. (This is not usually a problem, since you should probably use an alternative such as a usertag if your code is complex enough to need `strict`.)

The tag expands to the value returned by the enclosed code (i.e., printing to STDOUT or STDERR is useless).

```
[perl]
    $name    = $Values->{name};
    $browser = $Session->{browser};
    return "Hi, $name! How do you like your $browser?
[/perl]
```

HTML example:

```
<PRE mv=perl>
    $name    = $Values->{name};
    $browser = $Session->{browser};
    return "Hi, $name! How do you like your $browser?
</PRE>
```

Object references are available for most Interchange tags and functions, as well as direct references to Interchange session and configuration values.

Object	***Description***
$CGI−>{key}	Hash reference to raw submitted values
$CGI_array−>{key}	Arrays of submitted values
$Carts−>{cartname}	Direct reference to shopping carts
$Config−>{key}	Direct reference to $Vend::Cfg
$DbSearch−>array(@args)	Do a DB search and get results
$Document−>header()	Writes header lines
$Document−>send()	Writes to output
$Document−>write()	Writes to page
$Scratch−>{key}	Direct reference to scratch area

$Session−>{key}	Direct reference to session area
$Tag−>tagname(@args)	Call a tag as a routine (UserTag too!)
$TextSearch−>array(@args)	Do a text search and get results
$Values−>{key}	Direct reference to user form values
$Variable−>{key}	Config variables (same as $Config−>{Variable});
&HTML($html)	Same as $Document−>write($html);
&Log($msg)	Log to the error log

For full descriptions of these objects, see Interchange Perl Objects.

56.52.2.1. tables

This should ba a whitespace−separated list of database tables you want to make available within the Perl code.

If you wish to use database values in your Perl code, the tag must pre−open the table(s) you will be using. Here is an example using the products table:

```
[perl tables=products]
    my $cost = $Tag->data('products', 'our_cost', $Values->{code});
    $min_price = $cost * ( 1 + $min_margin );
    return ($min_price > $sale_price) ? $min_price : $sale_price;
[/perl]
```

If you do not do this, your code will fail with a runtime Safe error when it tries to look up 'our_cost' in the products database with the `data` tag.

Even if you properly specify the tables to pre−open, some database operations will still be restricted because `Safe` mode prohibits creation of new objects. For SQL, most operations can be performed if the `Safe::Hole` module is installed. Otherwise, you may have to set the `global=1` attribute to use data from SQL tables.

Interchange databases can always be accessed as long as they are pre−opened by using an item first.

Technical note:

Safe objects (including database handles) may persist within a page, and the `perl` tag does not necessarily destroy objects created earlier in the page. As a result, your code may work even though you did not set '`tables`' properly, only to break later when you change something elsewhere on the page.

For example, this will work because the first call to [`accessories` ...] opens the (default) products table:

```
[accessories code=os28044 attribute=size]

[perl]
  return $Tag->accessories( { attribute => 'size',
                              code      => 'os28085' } );
[/perl]
```

If you remove the first [accessories ...] tag, then the $Tag−>accessories call will fail with a Safe error unless you also set '`tables`=products' in the `perl` tag.

The moral of this story is to ensure that you pass all necessary tables in the `perl` tag.

56.52.2.2. failure

If your code contains a compile or runtime error and fails to evaluate (i.e., eval($code) would set $@), the tag will return the value set for the `failure` attribute. The error will be logged as usual.

For example,

```
[perl failure="It Broke"]
    my $cost = $Tag->data('products', 'our_cost', $Values->{code});
    $min_price = $cost * ( 1 + $min_margin );
    return ($min_price > $sale_price) ? $min_price : $sale_price;
[/perl]
```

will return 'It Broke' because the $Tag->Data(...) call will fail under the Safe module (see tables above).

56.52.2.3. no_return

If `no_return=1`, this attribute suppresses the return value of the perl code.

You can retrieve the return value from the session hash via `[data session mv_perl_result]` until it gets overwritten by another `perl` tag.

If `no_return` is set, the `perl` tag *will* return any output explicitly written with the &HTML or $Document->write() functions.

Note:

If `no_return` is *not* set, then the $Document->write() buffer is not returned (unless you use $Document->hot(1) or $Document->send(), in which case the contents of the write buffer will probably appear before anything else on the page). See Interchange Perl Objects for more detail.

Here is an example:

```
[perl tables=products no_return=1]
    my $cost = $Tag->data('products', 'our_cost', $Values->{code});
    $min_price = $cost * ( 1 + $min_margin );
    &HTML( ($min_price > $sale_price) ? $min_price : $sale_price );
    return ($min_price > $sale_price) ? 'too low' : 'ok';
[/perl]
```

This will put the same price on the page as our earlier example, but

`$Session->{mv_perl_result}` will be either 'too low' or 'ok'.

The [`mvasp`] tag is very similar to [perl no_return=1].

56.52.2.4. subs

If you have set the `AllowGlobal` catalog directive, setting `subs=1` will enable you to call GlobalSub routines within the enclosed perl code. Note that this can compromise security.

56.52.2.5. global

If you have set the `AllowGlobal` catalog directive, setting `global=1` will turn off Safe protection within the tag.

The code within the tag will then be able to do anything the user ID running Interchange can. This seriously compromises security, and you should know what you are doing before using it in a public site. It is especially dangerous if a single Interchange server is shared by multiple companies or user IDs.

Also, full '`use strict`' checking is turned on by default when in global mode. You can turn it off by using '`no strict;`' within your code. Note that any `strict` errors will go to the Interchange error logs, and the tag itself will fail silently within the page.

56.52.2.6. file

This prepends the contents of the specified file or FileDatabase entry to the enclosed perl code (if any), then executes as usual.

For example,

```
[perl file="my_script.pl"][/perl]
```

would execute myscript.pl and expand to its return value.

Absolute filenames (or filenames containing '../') are prohibited by the `NoAbsolute` catalog directive.

If the filename is not absolute, Interchange first looks for a file in the current directory, then in the list set with the `TemplateDir` catalog directive. If it fails to find a file by that name, it then looks for an entry by that name in the database specified with the `FileDatabase` catalog directive.

56.52.2.7. file

Add line numbers to the source code displayed in the error.log, amazingly useful if some of the perl is being generated elsewhere and interpolated.

56.52.2.8. eval_label

Set to a string, will replace the (eval ###) in the error message with this label, handy to quickly track down bugs when you have more than one perl block in the page, especially if you are using short_errors.

56.52.2.9. short_errors

If set to a true value, syntax errors and the like in perl tags will log just the error, not the whole source code of the block in question, handy when you have the code open in an editor anyway and don't want the error itself to get scrolled away when running 'tail −f error.log'.

56.52.2.10. trim_errors

If set to a number, and the error produced includes a line number, then only that number of lines before and after the broken line itself will be displayed, instead of the whole block.

56.53. price

56.53.1. Summary

Parameters: **code**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
    $Tag->price(
        {
         code => VALUE,
        }
    )

 OR

    $Tag->price($code, $ATTRHASH);
```

Attribute aliases

```
            base ==> mv_ib
```

56.53.2. Description

Arguments:

```
        code       Product code/SKU
        base       Only search in product table *base*
        quantity   Price for a quantity
        discount   If true(1), check discount coupons and apply
        noformat   If true(1), don't apply currency formatting
```

Expands into the price of the product identified by code as found in the products database. If there is more than one products file defined, they will be searched in order unless constrained by the optional argument **base**. The optional argument **quantity** selects an entry from the quantity price list. To receive a raw number, with no currency formatting, use the option `noformat=1`.

Interchange maintains a price in its database for every product. The price field is the one required field in the product database −− it is necessary to build the price routines.

For speed, Interchange builds the code that is used to determine a product's price at catalog configuration

time. If you choose to change a directive that affects product pricing you must reconfigure the catalog.

Quantity price breaks are configured by means of the *CommonAdjust* directive. There are a number of CommonAdjust recipes which can be used; the standard example in the demo calls for a separate pricing table called `pricing`. Observe the following:

```
CommonAdjust  pricing:q2,q5,q10,q25, ;products:price, ==size:pricing
```

This says to check quantity and find the applicable column in the pricing database and apply it. In this case, it would be:

```
2-4      Column *q2*
5-9      Column *q5*
10-24    Column *q10*
25 up    Column *q25*
```

What happens if quantity is one? It "falls back" to the price that is in the table `products`, column `price`.

After that, if there is a size attribute for the product, the column in the pricing database corresponding to that column is checked for additions or subtractions (or even percentage changes).

If you use this tag in the demo:

```
[price code=99-102 quantity=10 size=XL]
```

the price will be according to the `q10` column, adjusted by what is in the XL column. (The row is of course 99-102.) The following entry in pricing:

```
code    q2   q5   q10  q25  XL
99-102  10   9    8    7    .50
```

Would yield 8.50 for the price. Quantity of 10 in the `q10` column, with 50 cents added for extra large (XL).

Following are several examples based on the above entry as well as this the entry in the `products` table:

```
code    description   price    size
99-102  T-Shirt       10.00    S=Small, M=Medium, L=Large*, XL=Extra Large
```

NOTE: The examples below assume a US locale with 2 decimal places, use of commas to separate, and a dollar sign ($) as the currency formatting.

```
TAG                                          DISPLAYS
-----------------------------------------    --------
[price 99-102]                               $10.00
[price code="99-102"]                        $10.00
[price code="99-102" quantity=1]             $10.00
[price code="99-102" noformat=1]             10
[price code="99-102" quantity=5]             $9.00
[price code="99-102" quantity=5 size=XL]     $9.50
[price code="99-102" size=XL]                $10.50
[price code="99-102" size=XL noformat=1]     10.5
```

Product discounts for specific products, all products, or the entire order can be configured with the [discount ...] tag. Discounts are applied on a per-user basis -- you can gate the discount based on membership in a

club or other arbitrary means.

Adding [discount 99−102] $s * .9[/discount] deducts 10% from the price at checkout, but the price tag will not show that unless you add the discount=1 parameter.

```
[price code="99-102"]            -->   $10.00
[price code="99-102" discount=1] -->   $9.00
```

See *Product Discounts*.

56.54. process

This is a shortcut for the 'process' action, expanding to your catalog URL and session ID. It is analogous to the `area` tag, but is more limited. The following expansion is illuastrative:

```
   [process target=targetframe]
---
   http://www.here.com/cgi-bin/mycatalog/process.html?\
   id=6CZ2whqo" TARGET="targetframe
```

(the trailing backslash indicates continuation, i.e., the result should be only one line)

Note the mismatched quotes in the expansion. Your surrounding HTML should supply the containing quotes, like this:

```
   <A HREF="[process target=targetframe]">...
```

Alias: **process_target**

56.54.1. Summary

```
[process target secure]
[process target=targetframe secure=1 other_named_attributes]
```

Parameters	*Description*	*Default*
target	The target frame or window	*None*
secure	Boolean. If true (secure=1), the URL will link to your secure server.	*No*

Attributes	*Default*
interpolate (reparse)	*No*

Other_Charactreristics	
Invalidates cache	*No*
Container tag	*No*

Tag expansion example:

```
   [process targetframe 1]
---
   http://secure.here.com/cgi-bin/mycatalog/process.html?\
   id=6CZ2whqo" TARGET="targetframe
```

ASP−like Perl call:

```
$Tag->process(  { target => 'frametarget',
                  secure => 1, } );
```

or similarly with positional parameters,

```
$Tag->process($target, $secure, $attribute_hash_reference);
```

56.55. query

Passes SQL statements through to SQL databases, or allows SQL queries via Interchange's database abstraction into non−SQL databases and text files. The latter use may require Jochen Wiedmann's SQL Statement module (included with Bundle::Interchange from CPAN).

56.55.1. Summary

```
[query sql]
[query sql="SQL_query_text" other_named_attributes]
```

Parameters	*Description*	*Default*
sql	The SQL statement. • Passed directly through to an SQL database. • For a non−SQL table, the tag interprets your SQL first. See Jochen Wiedmann's SQL Statement module for limitations and detail.	*none*
query	Alias for `sql`	*none*

Attributes	*Default*
table	products
base (alias for `table`)	products
type (row_count, html, list, textref)	*none: uses arrayref="" if no type*
arrayref	*arrayref="" if no* `type` *given*
hashref	*none*
more (type=list)	*No*
xx form var. abbrev. (type=list)	*see form variable*
_ (type=list)	sql
list_prefix (type=list)	list
random (type=list)	*No*
safe_data (type=list)	*No*
label (type=list)	current
form (type=list)	*none*
wantarray	*No*
interpolate	*No*
reparse	*Yes*

Other_Charactreristics	
Invalidates cache	*No*

Container tag	*Yes*
Has subtags	*Yes*
Nests	*No*

Tag usage example:

This will list sku, description and price for ten products per page, followed by hyperlinks to the other pages of the list. Note that you may interpolate Interchange tags in the usual way if you double−quote the SQL statement.

```
[query sql="select sku, description, price from products where price < [value mv_arg]"
      type=list
      more=1
        ml=10]

   [on_match]Matched<br>[/on_match]
   [no_match]Not Found<br>[/no_match]

   [list]
     [sql−code] [sql−param description] [sql−price]
   [/list]

   [more_list]
     [more]
   [/more_list]
[/query]
```

ASP−like Perl call:

```
my $sql = "select * from products order by price";
my $result_array = $Tag->query( { sql => $sql,  },
                                $body );
my ($same_results, $col_name_hash, $col_name_array) =
                   $Tag->query( { sql => $sql,  },
                                $body );

my $result_hasharray = $Tag->query( { sql     => $sql,
                                      hashref => 'my_results',  },
                                    $body );
```

or similarly with positional parameters,

```
 $Tag->query( $sql, $attribute_hash_reference, $body);
```

56.55.2. Description

The `query` tag allows you to make SQL queries. If you are using an SQL database table, the tag will pass your SQL statement directly to the database and return the result.

If your table is not in an SQL database (for example, GDBM, text, LDAP, and in−memory tables), Interchange will internally convert it to an Interchange search specification with Jochen Wiedmann's SQL Statement module (included with Bundle::Interchange from CPAN). This means that you can use simple SQL queries regardless of the underlying database implementation.

56.55.2.1. Subtags

For list queries (type=list), the following subtags are available:

Subtag	*Usage*
on_match	[on_match] *do this if something matched* [/on_match]
no_match	[no_match] *do this if nothing matched* [/no_match]
list	[*list_prefix*] *do this for each matched item* [/*list_prefix*] The 'list' subtag defines a region where you can use any of the looping subtags that work in array−list context (see Looping tags and Sub−tags). The default looping tag prefix will be 'sql'. Note however that you can override this by setting the prefix attribute in the enclosing query tag. Similarly, the list_prefix attribute renames the [list] subtag itself to the value you set (see list_prefix below).
more_list	[more_list] [more] [/more_list] The 'more_list' and 'more' subtags are used when paginating the query results (see 'more' attribute). The [more] subtag will expand to a list of links to the other pages of the query results.

See also the example at the end of the Summary section above.

56.55.2.2. Perl and ASP usage

If you are calling $Tag−>query within a perl tag (or whenever the code is secured by the Safe.pm module), you must be sure to set the tables attribute properly in the enclosing perl tag (see the perl tag documentation for detail).

The types that return text to a page (i.e., row_count, html, and textref) work as usual, returning an appropriate string. Note that you may also have access to the results as an array reference in $Vend::Interpolate::Tmp−>{"} for the life of the page.

If you do not set a type, the tag will return a reference to an array of array references, since the default with no type is arrayref="".

If you call $Tag−>query in scalar context and set arrayref or hashref, it will return your results as a reference to an array of either arrayrefs or hashrefs, respectively (i.e., the same data structures you would get from Perl's DBI.pm module with fetchall_arrayref).

In list context, the first returned element is the aforementioned reference to your results. The second element is a hash reference to your column names, and the third element is an an array reference to the list of column names.

The following examples should be illustrative:

```
[perl tables=products]
  my $sql = "select sku, price, description from products
             where price < 10 order by price";

  my $results = $Tag->query( { sql => $sql, } );
  my ( $same_results, $col_name_hashref, $col_name_arrayref)
      = $Tag->query( { sql => $sql, } );

  my $hash_results = $Tag->query( {     sql => $sql,
                                    hashref => 'my_results' } );

  # $Vend::Interpolate::Tmp->{my_results} == $hash_results
  # $Vend::Interpolate::Tmp->{''} == $results == $same_results

  return $Tag->uneval( $results );
[/perl]
```

Technical Note: The $Tag->query() call works a bit differently in GlobalSubs and UserTags than within a perl tag. Specifically, in a GlobalSub or global UserTag, if you call query() in list context and want the three references (i.e., results, column hash and column array), then you need to set the 'wantarray=1' attribute in the query() call. See the wantarray attribute.

56.55.2.3. sql

This is the text of your SQL statement. The standard Interchange quoting rules apply. For example, use double quotes (") if you want to interpolate Interchange tags within your SQL statement, backticks (`) to calculate a value, *etc.*

```
[query sql="select description, price from products
            where price < [value mv_arg]" ...]
   ...
[/query]
```

56.55.2.4. table

The table attribute sets the database to use for the query. The default will typically be the database containing the '`products`' table (unless you have changed the first entry in $Vend::Cfg->{ProductFiles}).

56.55.2.5. type

If you are not setting the '`arrayref`' or '`hashref`' attributes, then the type attribute defines the way the query will return its results. The `type` should be one of the following:

Type	***Returns***

html	The html type returns the results in an html table. You will need to supply the enclosing <TABLE ...> and </TABLE> html tags. The following is an example of typical usage:
	
	`<TABLE>`
	`  [query sql="select * from products`
	`              where price > 12`
	`              order by price"`
	`        type=html]`
	`  [/query]`

	</TABLE>
list	This allows you to use subtags to control the query output and pagination. See the Subtags section above for detail.
row_count	This causes the tag to return the number of rows in the query result.

textref	This causes the tag to return a the query results as a serialized array of arrays that Perl can evaluate with its **eval()** function. Here is an illustrative example:
	
	`my $rows = eval( $Tag->query( { sql  => "select * from products"`
	`type => "textref" } )`
	`);`
	
	`my $r3_c0 = $rows->[3]->[0];`

If you do not specify a type, the tag will create an arrayref as if you had set 'arrayref="".

56.55.2.6. arrayref and hashref

If you set '`arrayref`=*keyname*' or '`hashref`=*keyname*', the query will not ruturn results to the page. Instead, it will place the results of your query in the $Vend::Interpolate::Tmp hash. Using '`arrayref`=my_query' sets $Vend::Interpolate::Tmp->{my_query} to refer to an array of array references, while '`hashref`=my_query' creates an array of hash references.

Note that this is useful only if you intend to access the results within Perl code (for example, within a [`perl`] tag), since there is no direct output to the returned page.

The $Vend::Interpolate::Tmp hash persists only for the life of the template page being processed. If you need the query results array reference to outlive the page, you will have to save the reference somewhere more persistent such as the $Session hash:

```
$Session->{my_query} = $Vend::Interpolate::Tmp->{my_query};
```

Beware the impact on perfomance if you do this with large result sets.

Technical note -- the string returned by the '`textref`' `type` will **eval**() to the '`arrayref`' data structure.

56.55.2.7. more

Requires '`type`=list'.

You must set `more`=1 to properly paginate your results from `list` queries (see '`type`=list' above. If you do not set `more`=1, then the links to later pages will merely redisplay the first page of your results.

56.55.2.8. *form variable abbreviations*

Requires '`type`=list'.

See the Search and Form Variables appendix for a list of form variables. Note that you must use the two-letter abbreviation rather than the full form variable name.

A few deserve special mention:

Abbr	*Name*	*Description*
ml	mv_matchlimit	Sets number of rows to return. If paginating (more=1), sets rows returned per page.
fm	mv_first_match	Start displaying search at specified match
sp	mv_search_page	Sets the page for search display
st	mv_searchtype	Forces a specific search type (text, glimpse, db or sql), overriding the default determined from your database implementation.

56.55.2.9.

Requires 'type=list'.

Setting 'prefix=foo' overrides the default prefix of 'sql' for loop subtags within a list region (see Looping tags and Sub−tags).

See the list_prefix attribute below for an illustrative example.

56.55.2.10. list_prefix

Requires 'type=list'.

Setting 'list_prefix=bar' overrides the default region tagname of 'list'. The best way to show this is by example. Compare the following two examples of list queries, the first using the defaults and the second with explicitly set prefix and list_prefix.

```
   [query sql="select sku, description, price from products
               where price < 20"
         type=list
         more=1
           ml=10]

      [on_match]Matched<br>[/on_match]
      [no_match]Not Found<br>[/no_match]

      [list]
        [sql-code] [sql-param description] [sql-price]
      [/list]

      [more_list]
        [more]
      [/more_list]
   [/query]

------------------------------------------------------------

   [query  sql="select sku, description, price from products
                where price < 20"
          type=list
        prefix=foo
   list_prefix=bar
          more=1
            ml=10]

      [on_match]Matched<br>[/on_match]
      [no_match]Not Found<br>[/no_match]

      [bar]
```

```
      [foo-code] [foo-param description] [foo-price]
    [/bar]

    [more_list]
      [more]
    [/more_list]
  [/query]
```

56.55.2.11. random

Requires '[type](#)=list'.

You can use the '`random`' attribute to randomly select a set of rows from the whole result set of your query. In other words, setting '`random`=*n*', where *n* > 0, causes the [`list`] region to loop over *n* randomly chosen rows rather than the full query result set.

The example below would display three randomly chosen products priced under 20.

```
[query sql="select * from products
            where price < 20"
      type=list
     random=3]

   [list]
     [sql-code] [sql-param description] [sql-price]
   [/list]

[/query]
```

56.55.2.12. safe_data

Requires '[type](#)=list'.

Note -- you should not set this unless you need it and know what you are doing.

Setting '`safe_data`=1' allows the [sql-data] tag to return values containing the '[' character. See also [Looping tags and Sub-tags](#).

Beware of reparsing issues.

56.55.2.13. label

Requires '[type](#)=list'.

If you are setting up multiple simultaneously active search objects within a page, this allows you to distinguish them. The default label is 'current'. Most people will not need this.

56.55.2.14. form

Requires '[type](#)=list'.

You can use this to pass one CGI form variable in the pagination links of a [more-list]. For example, '`form`="foo=bar"' to include '&foo=bar' in the URL of each of the pagination links.

Note that the variable will not be available in the initial result set since the query returns the first page directly (i.e., you did not follow a pagination link).

56.55.2.15. wantarray

This is relevant only when calling $Tag−>query(...) within global Perl code such as a globalsub or global usertag where $MVSAFE::Safe is not defined. In these cases, setting '`wantarray`=1' allows the call to

```
$Tag−>query( { wantarray => 1, ... }, ... );
```

to return references as it would if called within an ordinary [`perl`] tag. Note that it does not force list context if you call $Tag−>query in scalar context.

Technical note −− the ordinary [`query` ...] ... [`/query`] usage forces scalar context on the query call and suppresses the return value for those `types` that would return references if $Tag−>query were called within a [`perl`] tag. The wantarray option is needed because global subs and usertags are also affected by this unless you set wantarray.

56.56. read_cookie

Returns the value of the named cookie. Returns nothing if the cookie does not exist.

56.56.1. Summary

```
[read_cookie name]
[read_cookie name=mycookie]
```

Attributes	*Description*	*Default*
name	The name of the cookie whose value you want	*none*

Attributes	*Default*
interpolate (reparse)	*No*

Other_Charactreristics	
Invalidates cache	*Yes*
Container tag	*No*

Usage example:

```
   [read−cookie name=MV_SESSION_ID]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
   6CZ2whqo
```

ASP−like Perl call:

```
$Tag−>read_cookie(  { name => $name, }  );
```

or similarly with positional parameters,

```
$Tag−>read_cookie( $name );
```

56.56.2. Description

This tag expands to the value of the named cookie (or nothing if the cookie does not exist).

See the Netscape specification at http://www.netscape.com/newsref/std/cookie_spec.html if you need more cookie−specific detail.

56.56.2.1. name

This is the name of the cookie whose value you want to retrieve.

56.56.2.2. Parsing an HTTP_COOKIE string

If you pass this tag a second parameter within a Perl call, it will use your value as the HTTP_COOKIE string (ignoring the real one). This only applies if you pass the values positionally within a perl call since there is no name for the HTTP_COOKIE string input:

```
$Tag->read_cookie('MV_SESSION_ID', "MV_SESSION_ID=UnHyaDQj:127.0.0.1; ...");
```

56.57. restrict

Restrict tag execution in a region. If a restricted tag is encountered, it is simply output.

56.57.1. Summary

```
[restrict tag1 tag2]
[restrict policy=deny enable="page area value"]
```

Attributes	*Description*	*Default*
policy	Whether to allow or deny by default.	**deny**
enable	Tags to enable when default policy is deny.	*none*
disable	Tags to disable. Overrides enable.	*none*

Attributes	*Default*
interpolate (reparse)	*No*

Other_Charactreristics	
Invalidates cache	*Yes*
Container tag	*No*

Usage example:

```
   [read-cookie name=MV_SESSION_ID]
-------------------------------------------------------
   6CZ2whqo
```

ASP−like Perl call:

```
    N/A. Cannot be called effectively.
```

56.57.2. Description

Restrict tag execution in a region. If a restricted tag is encountered, it is simply output. It can be used to allow certain tags in a user−editable region, while denying dangerous tags. Or it can be used to restrict all tag execution in a region.

56.57.2.1. policy

Default is `deny`, which makes most sense. You then specifically enable certain ITL tags. If you set `allow` by default, you must be very careful that you really are disabling all of what you consider to be dangerous tags.

56.57.2.2. enable

A space−separated or comma−separated list of tags to disable when the default policy is `deny`. Has no effect when the default policy is `allow`, and any tags passed in the disable parameter override the enable.

56.57.2.3. disable

A space−separated or comma−separated list of tags to disable when the default policy is `allow`. If you have a list of tags that are enabled, perhaps stored in a scratch variable, you can disable some of those tags since this takes precedence over the enable.

56.58. row

56.58.1. Summary

Parameters: **width**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [row] FOO [/row]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
    $Tag->row(
        {
         width => VALUE,
        },
        BODY
    )

OR
```

```
$Tag->row($width, $BODY);
```

56.58.2. Description

Formats text in tables. Intended for use in emailed reports or <PRE></PRE> HTML areas. The parameter *nn* gives the number of columns to use. Inside the row tag, [col param=value ...] tags may be used.

56.58.2.1. [col width=nn wrap=yes|no gutter=n align=left|right|input spacing=n]

Sets up a column for use in a [row]. This parameter can only be contained inside a [row nn] [/row] tag pair. Any number of columns (that fit within the size of the row) can be defined.

The parameters are:

```
width=nn        The column width, I<including the gutter>. Must be
                supplied, there is no default. A shorthand method
                is to just supply the number as the I<first> parameter,
                as in [col 20].

gutter=n        The number of spaces used to separate the column (on
                the right-hand side) from the next. Default is 2.

spacing=n       The line spacing used for wrapped text. Default is 1,
                or single-spaced.

wrap=(yes|no)   Determines whether text that is greater in length than
                the column width will be wrapped to the next line. Default
                is I<yes>.

align=(L|R|I)   Determines whether text is aligned to the left (the default),
                the right, or in a way that might display an HTML text
                input field correctly.
```

56.58.2.2. [/col]

Terminates the column field.

56.59. salestax

56.59.1. Summary

Parameters: **name noformat**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->salestax(
        {
         name => VALUE,
         noformat => VALUE,
        }
    )

 OR

    $Tag->salestax($name, $noformat);
```

Attribute aliases

```
            cart ==> name
```

56.59.2. Description

Expands into the sales tax on the subtotal of all the items ordered so far for the cart, default cart is `main`. If there is no key field to derive the proper percentage, such as state or zip code, it is set to 0. If the noformat tag is present and non−zero, the raw number with no currency formatting will be given.

56.60. scratch

56.60.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->scratch(
        {
         name => VALUE,
        }
    )

 OR

    $Tag->scratch($name);
```

56.60.2. Description

Returns the contents of a scratch variable to the page. (A scratch variable is set with a [set] value [/set] container pair.)

56.61. scratchd

Deletes the named scratch variable and returns its value before the deletion. For example,

```
[scratchd varname_to_delete]
```

deletes the scratch variable *varname_to_delete*.

See also the `scratch` and `set` tags.

56.61.1. Summary

```
[scratchd P_PARAM]
[scratchd N_PARAM other_named_attributes]
```

Parameters	***Description***	***Default***
name	Name of scratch variable to delete	*None*

Attributes	***Default***
interpolate (reparse)	*No*

Other_Charactreristics	
Invalidates cache	*Yes*
Container tag	*No*

Tag expansion example:

```
   [set myvar]This is myvar[/set]
   .
   .
   .
   [scratchd myvar]
---
   This is myvar
```

ASP−like Perl call:

```
    $Tag->scratchd($name, $attribute_hash_reference);
```

56.61.2. Description

Deletes the named scratch variable and returns its value before the deletion.

56.62. search_list

Formats results returned by a search. Must be enclosed within a `search region`. Has sub−tags (see `Looping tags and Sub−tags`).

56.63. search_region

56.63.1. Summary

Parameters: **arg**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [search_region] FOO [/search_region]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
    $Tag->search_region(
        {
         arg => VALUE,
        },
        BODY
    )

 OR

    $Tag->search_region($arg, $ATTRHASH, $BODY);
```

Attribute aliases

```
            args ==> arg
            params ==> arg
            search ==> arg
```

56.63.2. Description

NO Description

56.64. selected

56.64.1. Summary

Parameters: **name value**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->selected(
        {
         name => VALUE,
         value => VALUE,
        }
    )

OR

    $Tag->selected($name, $value, $ATTRHASH);
```

56.64.2. Description

You can provide a "memory" for drop−down menus, radio buttons, and checkboxes with the [checked] and [selected] tags.

This will output SELECTED if the variable `var_name` is equal to `value`. If the optional MULTIPLE argument is present, it will look for any of a variety of values. Not case sensitive unless the optional `case=1` parameter is used.

Here is a drop−down menu that remembers an item−modifier color selection:

```
    <SELECT NAME="color">
    <OPTION [selected color blue]> Blue
    <OPTION [selected color green]> Green
    <OPTION [selected color red]> Red
    </SELECT>
```

Here is the same thing, but for a shopping−basket color selection

```
    <SELECT NAME="[modifier-name color]">
    <OPTION [selected [modifier-name color] blue]> Blue
    <OPTION [selected [modifier-name color] green]> Green
    <OPTION [selected [modifier-name color] red]> Red
    </SELECT>
```

By default, the Values space (i.e. [value foo]) is checked −− if you want to use the volatile CGI space (i.e. [cgi foo]) use the option `cgi=1`.

56.65. set

56.65.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [set] FOO [/set]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->set(
        {
         name => VALUE,
        },
        BODY
    )

OR

    $Tag->set($name, $BODY);
```

56.65.2. Description

Sets a scratch variable to *value*.

Most of the mv_* variables that are used for search and order conditionals are in another namespace −− they can be set by means of hidden fields in a form.

You can set an order profile with:

```
  [set checkout]
  name=required
  address=required
  [/set]
  <INPUT TYPE=hidden NAME=mv_order_profile VALUE="checkout">
```

A search profile would be set with:

```
  [set substring_case]
  mv_substring_match=yes
  mv_case=yes
  [/set]
  <INPUT TYPE=hidden NAME=mv_profile VALUE="substring_case">
```

Any of these profile values can be set in the OrderProfile files as well.

56.66. set_cookie

Sets browser cookie(s) with the specified attributes.

56.66.1. Summary

```
[set_cookie named_attributes]
```

Parameters must be named (no positional usage except in Perl call)

Attributes	*Description*	*Default*
name	The name you give the cookie	*none*
value	The value (automatically html−escaped by Interchange)	*none*
expire	Expiration date as "Wdy, DD−Mon−YYYY HH:MM:SS GMT"	*none*
domain	Overrides the domain(s) set in `CookieDomain`	*Domain(s), if any, defined in the* `CookieDomain` *directive*
path	legal URL paths for the cookie	*URL path(s) to your catalog, including aliases*

Other_Charactreristics	
Invalidates cache	*Yes*
Container tag	*No*

Usage example:

```
   [set−cookie name=mycookie
              value="the value"
             expire="Tue, 03−Apr−2001 17:00:00 GMT" ]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
   This tag returns no value in the page
```

ASP−like Perl call:

```
    $Tag−>set_cookie(  { name   => $name,
                         value  => $value,
                         expire => $expire,
                         domain => $domain,
                         path   => $path, }  );
```

or similarly with positional parameters,

```
    $Tag−>set_cookie( $name, $value, $expire, $domain, $path );
```

56.66.2. Description

This tag sets one or more browser cookies with your specified name, value, and expiration. (Interchange will set more than one cookie if needed to ensure that the cookie is visible from all `Catalog` URL path aliases and `CookieDomains`.)

See the Netscape specification at http://www.netscape.com/newsref/std/cookie_spec.html for more cookie−specific detail.

If you need access to the cookie from outside of your Interchange catalog, you can also set the domain and URL paths for which the cookie will be valid. If you need the cookie only within your catalog and the domains specified by the `CookieDomain` directive, you probably should not override the Interchange domain and path defaults.

56.66.2.1. name

This is the name of the cookie. This is the key you will use when reading the cookie later.

56.66.2.2. value

This is the value to store in the cookie.

56.66.2.3. expire

Persistent cookies (that outlive a browser session) require an expiration date. The date must be a string of the form:

"Wdy, DD−Mon−YYYY HH:MM:SS GMT"

and the timezone must be GMT.

If you do not supply a date, the cookie will disappear when the user closes the browser.

56.66.2.4. domain

The value you set will override the Interchange default domain(s). You might set this if you need access to the cookie from outside the Interchange catalog, but it is usually better to set the `CookieDomain` directive in your catalog.

The default is to use your catalog's domain or all `CookieDomain` values.

56.66.2.5. path

The value you set will override the Interchange default URL path(s).

The default is to set a cookie with a path for each catalog alias (see the `Catalog` directive). This ensures that the cookie will be visible regardless of how the end user returns to your catalog.

56.67. seti

56.67.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [seti] FOO [/seti]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->seti(
        {
         name => VALUE,
        },
        BODY
    )

OR

    $Tag->seti($name, $BODY);
```

56.67.2. Description

Equivalent to the [set] tag, except that it interpolates by default.

56.68. setlocale

Sets locale and/or currency for the current page. Can be made persistent for the user with the 'persist' option. Resets default locale if called without arguments. See also Setting the Locale in the template documentation.

56.68.1. Summary

Parameters: locale currency

- locale
  - The locale to use.
  - Default: [scratch mv_locale] (see also 'persist' attribute)
- currency
  - The currency format to use.
  - Default: [scratch mv_currency] (see also 'persist' attribute)

Positional parameters in same order.

Named Attrbutes

- persist
  - Setting '`persist=1`' also sets the scratch variables, **mv_locale** and **mv_currency** to specified locale and currency. This makes the locale settings persistent for the user's session. Otherwise (`persist=0`), the `setlocale` tag affects the remainder of the current page

only.

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
    $Tag->setlocale(
        {
         locale => VALUE,
         currency => VALUE,
        }
    )

OR

    $Tag->setlocale($locale, $currency, $ATTRHASH);
```

56.68.2. Description

Immediately sets the locale to `locale`, and will cause it to persist in future user pages if the `persist` is set to a non−zero, non−blank value. If the `currency` attribute is set, the pricing and currency−specific locale keys and Interchange configuration directives are modified to that locale. If there are no arguments, it sets it back to the user's default locale as defined in the scratch variables `mv_locale` and `mv_currency`.

This allows:

```
    Dollar Pricing:

    [setlocale en_US]
    [item-list]
    [item-code]: [item-price]<BR>
    [/item-list]

    Franc Pricing:

    [setlocale fr_FR]
    [item-list]
    [item-code]: [item-price]<BR>
    [/item-list]

    [comment] Return to the user's default locale [/comment]
    [setlocale]
```

56.69. shipping

Returns the cost of shipping the items in the cart via the shipping mode set with the **mode** parameter. See also the Shipping section of the Database documentation.

56.69.1. Summary

Parameters: **mode**

- mode
  - Aliases: **name**, **modes**
  - Whitespace, null or comma delimited list of modes for which to calculate shipping cost. See also mv_shipmode.
  - Default: [value mv_handling] if **handling=1** or [value mv_shipmode] or 'default'
- table
  - Alias: **tables**
  - Whitespace−delimited list of tables containing shipping data required for perl or query calculations (*e.g.*, in the 'PERL' field of your shipping database −− see Shipping). You must specify the tables to get past the Perl 'Safe.pm' proctection. For example, you will get 'Safe' errors if you refer to an SQL table without specifying it here.
  - Default: None
- cart
  - Alias: **carts**
  - Comma−delimited list of names of carts to calculate shipping cost for.
  - Default: current cart
- convert
  - Applies the conversion (if any) set with the `PriceDivide` catalog configuration directive.
  - Default: True
- noformat
  - Returns result as a number rather than a string formatted for the current locale's currency.
  - Default: True
- handling
  - Boolean−− use [value mv_handling] rather than [value mv_shipping] as first default for **mode**. Note that this attribute matters only if you do not specify the **mode** in the tag.
  - Note that this is set by the [`handling` tag (which calls the `shipping` tag internally). You should probably use the handling tag rather than setting this directly.
  - Default: False
- reset_modes
  - Clears list of modes in $Vend::Cfg−>{Shipping_line}
  - Default: False
- add
  - Adds the argument to **add** as data for a shipping.asc file (in $Vend::Cfg−>{ScratchDir}/) and sets it.
- file
  - Filename to read shipping from (default is usual shipping database, e.g., shipping.asc)
- label
  - By default, returns HTML <OPTION ...> widget for shipping mode(s), including description and cost. You can override the widget with the **format** attribute. Note that **label** overrides **noformat**.
  - Here is an example from the foundation checkout.html page:

```
[shipping
  label=1
  mode=|[data table=country key='[default country US]' col=shipmodes]|
]
```

- format
  - Format for results with **label** attribute.
  - Default: '<OPTION VALUE="%M"%S>%D (%F)'
  - For example,
    [shipping mode="FLAT"
    label=1
    format="My Format Desc %D Price %F"]
- default
  - Resets shipping mode to default of [value mv_shipmode]
- hide
  - Suppresses output
- reset_message
  - Boolean. Blanks the session's current shipping message (i.e., $Session−>{ship_message}).

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->shipping(
        {
         mode => VALUE,
        }
    )

OR

    $Tag->shipping($mode, $ATTRHASH);
```

Attribute aliases

```
carts ==> cart
modes ==> mode
name ==> mode
tables ==> table
```

56.69.2. Description

This tag calculates the shipping cost for items in the current cart via the specified shipping mode (usually set in the mv_shipmode variable). See the Shipping section of the Database documentation for detail.

56.69.2.1. Rounding

Note that the tag rounds the calculated shipping cost to a locale−specific number of fractional digits (e.g., to the nearest penny, or 2 digits after the decimal point in the USA).

56.70. shipping_desc

Returns the shipping description for the specified shipping **mode**. See the Shipping section of the Database

documentation. See also shipping.asc database for shipping modes.

Alias: **shipping_description**

The two tags below are identical in operation:

```
[shipping_desc mode]
[shipping_description mode]
```

56.70.1. Summary

```
[shipping_desc mode]
[shipping_desc mode=shipping_mode]
```

Parameters	*Description*	*Default*
mode	Shipping mode. This is a key into the shipping.asc database. See Shipping documentation.	`mv_shipmode`, or 'default' if `mv_shipmode` not set

Other_Charactreristics	
Invalidates cache	Yes, but only if no **mode** given
Container tag	*No*

Tag expansion example:

```
[shipping_desc 1DM]
---
UPS Next Day Air Early AM
```

ASP−like Perl call:

```
$Tag->shipping_desc( $mode );
```

56.71. soap

56.71.1. Summary

Parameters: **call uri proxy**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
$Tag->soap(
```

```
        {
         call => VALUE,
         uri => VALUE,
         proxy => VALUE,
        }
    )

OR

    $Tag->soap($call, $uri, $proxy, $ATTRHASH);
```

56.71.2. Description

NO Description

56.72. strip

Strips leading and trailing whitespace from the contained body text.

56.72.1. Summary

```
[strip]
   Body text to strip
[/strip]
```

No parameters.

Other_Charactreristics	
Invalidates cache	*No*
Container tag	*Yes*
Has Subtags	*No*

ASP−like Perl call:

```
$Tag->strip($BODY);
```

or even better, just do it directly like this

```
$BODY =~ s/^\s+//;
$BODY =~ s/\s+$//;
```

56.73. subtotal

56.73.1. Summary

Parameters: **name noformat**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->subtotal(
        {
         name => VALUE,
         noformat => VALUE,
        }
    )

OR

    $Tag->subtotal($name, $noformat);
```

Attribute aliases

```
            cart ==> name
```

56.73.2. Description

Positional: [subtotal cart* noformat*]

mandatory: NONE

optional: cart noformat

Expands into the subtotal cost, exclusive of sales tax, of all the items ordered so far for the optional `cart`. If the noformat tag is present and non−zero, the raw number with no currency formatting will be given.

56.74. tag

56.74.1. Summary

Parameters: **op arg**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [tag] FOO [/tag]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
    $Tag->tag(
        {
         op => VALUE,
         arg => VALUE,
        },
        BODY
    )

OR

    $Tag->tag($op, $arg, $ATTRHASH, $BODY);
```

Attribute aliases

```
            description ==> arg
```

56.74.2. Description

Performs any of a number of operations, based on the presence of `arg`. The arguments that may be given are:

56.74.2.1. export database file* type*

Exports a complete Interchange database to its text source file (or any specified file). The integer `n`, if specified, will select export in one of the enumerated Interchange export formats. The following tag will export the products database to products.txt (or whatever you have defined its source file as), in the format specified by the *Database* directive:

```
    [tag export products][/tag]
```

Same thing, except to the file products/new_products.txt:

```
    [tag export products products/newproducts.txt][/tag]
```

Same thing, except the export is done with a PIPE delimiter:

```
    [tag export products products/newproducts.txt 5][/tag]
```

The file is relative to the catalog directory, and only may be an absolute path name if *NoAbsolute* is set to `No`.

56.74.2.2. flag arg

Sets an Interchange condition.

The following enables writes on the `products` and `sizes` databases held in Interchange internal DBM format:

```
    [tag flag write]products sizes[/tag]
```

SQL databases are always writable if allowed by the SQL database itself −− in−memory databases will never be written.

The [tag flag build][/tag] combination forces static build of a page, even if dynamic elements are contained. Similarly, the [tag flag cache][/tag] forces search or page caching (not usually wise).

56.74.2.3. log dir/file

Logs a message to a file, fully interpolated for Interchange tags. The following tag will send every item code and description in the user's shopping cart to the file logs/transactions.txt:

```
[tag log logs/transactions.txt]
[item_list][item-code]  [item-description]
[/item_list][/tag]
```

The file is relative to the catalog directory, and only may be an absolute path name if *NoAbsolute* is set to `No`.

56.74.2.4. mime description_string

Returns a MIME-encapsulated message with the boundary as employed in the other mime tags, and the `description_string` used as the Content-Description. For example

```
[tag mime My Plain Text]Your message here.[/tag]
```

will return

```
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-ID: [sequential, lead as in mime boundary]
Content-Description: My Plain Text

Your message here.
```

When used in concert with [tag mime boundary], [tag mime header], and [tag mime id], allows MIME attachments to be included -- typically with PGP-encrypted credit card numbers. See the demo page ord/report.html for an example.

56.74.2.5. mime boundary

Returns a MIME message boundary with unique string keyed on session ID, page count, and time.

56.74.2.6. mime header

Returns a MIME message header with the proper boundary for that session ID, page count, and time.

56.74.2.7. mime id

Returns a MIME message id with the proper boundary for that session ID, page count, and time.

56.74.2.8. show_tags

The encased text will not be substituted for with Interchange tags, with < and [characters changed to &#lt; and [respectively.

```
[tag show_tags][value whatever][/tag]
```

56.74.2.9. time

Formats the current time according to POSIX strftime arguments. The following is the string for Thursday, April 30, 1997.

```
[tag time]%A, %B %d, %Y[/tag]
```

56.74.2.10. touch

Touches a database to allow use of the tag_data() routine in user−defined subroutines. If this is not done, the routine will error out if the database has not previously been accessed on the page.

```
[tag touch products][/tag]
```

56.75. time

56.75.1. Summary

Parameters: **locale**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [time] FOO [/time]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
$Tag->time(
    {
     locale => VALUE,
    },
    BODY
)
```

OR

```
$Tag->time($locale, $ATTRHASH, $BODY);
```

56.75.2. Description

Formats the current time according to POSIX strftime arguments. The following is the string for Monday, January 1, 2001.

```
[time]%A, %B %d, %Y[/tag]
```

See the `strftime` man page for information on the arguments (which are the same as modern UNIX date commands).

Accepts the following options:

56.75.2.1. adjust

If you wish to temporarily adjust the time for display purposes, you can pass an `adjust` parameter with the number of hours (plus or minus) from the local time or from GMT:

```
[time]%c[/time]
[time adjust="-3"]%c[/time]
```

Will display:

```
Mon 01 Jan 2001 11:29:03 AM EST
Mon 01 Jan 2001 08:29:03 AM EST
```

Note that the time zone does not change -- you should either pick a format which doesn't display zone, use the `tz` parameter, or manage it yourself.

NOTE: You can adjust time globally for an Interchange installation by setting the $ENV{TZ} variable on many systems. Set TZ in your environment and then restart Interchange:

```
## bash/ksh/sh
TZ=PST7PDT; export TZ
interchange -restart

## csh/tcsh
setenv TZ PST7PDT
interchange -restart
```

On most modern UNIX systems, all times will now be in the PST zone.

56.75.2.2. gmt

If you want to display time as GMT, use the `gmt` parameter:

```
[time]%c[/time]
[time gmt=1]%c[/time]
```

will display:

```
Mon 01 Jan 2001 11:33:26 AM EST
Mon 01 Jan 2001 04:33:26 PM EST
```

Once again, the zone will not be set to GMT, so you should pick a format string which doesn't use zone, use the `tz` parameter, or manage it yourself.

56.75.2.3. locale

Format the time according to the named `locale`, assuming that locale is available on your operating system. For example, the following:

```
[time locale=en_US]%B %d, %Y[/time]
[time locale=fr_FR]%B %d, %Y[/time]
```

should display:

```
January 01, 2001
janvier 01, 2001
```

56.75.2.4. tz

Use the passed `tz` to display the time. Will adjust for hours difference.

Example:

```
[time tz=GMT0]
[time tz=CST6CDT]
[time tz=PST8PDT]
```

will display:

```
Mon 01 Jan 2001 04:43:02 PM GMT
Mon 01 Jan 2001 10:43:02 AM CST
Mon 01 Jan 2001 08:43:02 AM PST
```

Note that the first alphabetical string is the zone name when not under daylight savings time, the digit is the number of hours displacement from GMT, and the second alphabetical string is the zone name when in daylight savings time. NOTE: This may not work on all operating systems.

56.75.2.5. zerofix

Strips leading zeroes from numbers.

56.76. timed_build

56.76.1. Summary

Parameters: **file**

Positional parameters in same order.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [timed_build] FOO [/timed_build]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
    $Tag->timed_build(
        {
         file => VALUE,
        },
        BODY
    )

OR

    $Tag->timed_build($file, $ATTRHASH, $BODY);
```

56.76.2. Description

Allows you to build CPU−intensive regions of ITL tags on a timed basis.

In the simplest case, surround a region of ITL with [timed−build] and [/timed−build]:

```
        [timed-build]
                Built at [time]%c[/time].
        [/timed-build]
```

If a `file` parameter is not passed, saves to the directory *timed* in catalog root, with the file name of the current page. If the `minutes` parameter is not passed specifying how often the page should be rebuilt, then it will not be rebuilt until the output file is removed.

Accepts the following parameters:

56.76.2.1. auto

Turns on automatic region processing. The text of the timed−build region is processed to determine a unique checksum or digest (using MD5), and that file name is checked in the directory tmp/auto−timed (assuming `ScratchDir` is set to `tmp`). If no number of minutes is supplied, `60` is assumed.

This is designed to automatically build regions of commonly used areas without having to manage the file name yourself.

Implies `login=1`, but will still abort if no session ID cookie has been sent. Use `force=1` to ignore cookie status.

56.76.2.2. file

Name of the file to save the results in. Relative to catalog root. The directory must exist.

56.76.2.3. if

Allows you to to only display the cached region when the `if` paremeter is true. For example, you can do:

```
        [timed-build if="[value timed]"]
        ITL tags....
        [/timed-build]
```

The cached region will only be displayed if the variable `timed` is set to a non−zero, non−blank value. Otherwise, the ITL tags will be re−interpreted every time.

56.76.2.4. minutes

The number of minutes after which the timed build should be repeated. If set to 0, it will be built once and then not rebuilt until the output file is removed.

56.76.2.5. period

Alternative way of expressing time. You can pass a string describing the rebuild time period:

```
[timed-build period="4 hours"]
ITL tags....
[/timed-build]
```

This is really the same as `minutes=240`. Useful for passing seconds:

```
[timed-build period="5 seconds"]
ITL tags....
[/timed-build]
```

The region will be rebuilt every 5 seconds.

Performance Tip: use minutes of .08 instead; avoids having to parse the period string.

If you have the StaticDir catalog.cfg parameter set to a writable path, you can build a cached static version of your catalog over time. Simply place a [timed−build] tag at the top of pages you wish to build statically. Assuming the catalog is not busy and write lock can be obtained, the StaticDBM database will be updated to mark the page as static and the next time a link is made for that page the static version will be presented.

56.77. tmp

56.77.1. Summary

Parameters: **name**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Interpolates **container text** by default>.

This is a container tag, i.e. [tmp] FOO [/tmp]. Nesting: NO

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
$Tag->tmp(
    {
     name => VALUE,
    },
```

```
        BODY
    )

OR

    $Tag->tmp($name, $BODY);
```

56.77.2. Description

Sets a scratch variable to *value*, but at the end of the user session the Scratch key is deleted. This saves session write time in many cases.

This tag interpolates automatically. (Interpolation can be turned off with `interpolate=0`.)

IMPORTANT NOTE: the [tmp ...][/tmp] tag is not appropriate for setting order profiles or `mv_click` actions. If you want to avoid that, use a profile stored via the catalog.cfg directive `OrderProfile`.

56.78. total_cost

56.78.1. Summary

Parameters: **name noformat**

Positional parameters in same order.

Pass attribute hash as last to subroutine: **no**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->total_cost(
        {
         name => VALUE,
         noformat => VALUE,
        }
    )

OR

    $Tag->total_cost($name, $noformat);
```

Attribute aliases

```
            cart ==> name
```

56.78.2. Description

Expands into the total cost of all the items in the current shopping cart, including sales tax (if any).

56.79. tree

56.79.1. Summary

Parameters: **table master subordinate start**

ONLY THE PARAMETERS ARE POSITIONAL.

The attribute hash reference is passed after the parameters but before the container text argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

This is a container tag, i.e. [tree] FOO [/tree]. Nesting: NO

Invalidates cache: **no**

Called Routine:

ASP−like Perl call:

```
    $Tag->tree(
        {
         table => VALUE,
         master => VALUE,
         subordinate => VALUE,
         start => VALUE,
        },
        BODY
    )

 OR

    $Tag->tree($table, $master, $subordinate, $start, $ATTRHASH, $BODY);
```

Attribute aliases

```
            sub ==> subordinate
```

56.79.2. Description

Provides iterative list capability for binary trees. It produces hash−based rows use the same tags as [item−list]; sets some additional hash key entries to describe the tree and provide display control.

Works on a data set with the structure:

```
    parent  child
    99      a
```

```
a       b
a       c
a       d
a       x
x       y
x       z
99      m
99      n
99      o
o       e
o       f
o       g
```

Sets several keys which assist in walking and displaying the tree.

56.79.2.1. mv_level

Level of the item. If it is in the first level, it is 0. Sublevels are infinite (except for performance).

56.79.2.2. mv_increment

Increment label for the item. Normally goes from 1...n, but can be changed to A...Z or a...z in outline mode.

56.79.2.3. mv_children

If in autodetect mode, set to the number of children this branch has. If a leaf, set to 0.

56.79.2.4. mv_spacing

A multiple of level times the spacing option. Useful for setting width of spacer images.

The above sample data would produce:

```
a           mv_level=0, mv_increment=1, mv_children=4
    b       mv_level=1, mv_increment=1, mv_children=0
    c       mv_level=1, mv_increment=2, mv_children=0
    d       mv_level=1, mv_increment=3, mv_children=0
    x       mv_level=1, mv_increment=4, mv_children=2
        y   mv_level=2, mv_increment=1, mv_children=0
        z   mv_level=2, mv_increment=2, mv_children=0
m           mv_level=0, mv_increment=1, mv_children=0
n           mv_level=0, mv_increment=2, mv_children=0
o           mv_level=0, mv_increment=3, mv_children=3
    e       mv_level=1, mv_increment=1, mv_children=0
    f       mv_level=1, mv_increment=2, mv_children=0
    g       mv_level=1, mv_increment=3, mv_children=0
```

from the tag call:

```
    <table>
[tree   start=99
        master=parent_fld
        subordinate=child_fld
        autodetect=1
                spacing=4
        full=1]
    <tr>
```

```
<td>
[if-item-param mv_level]
        [item-calc]
                return ' ' x [item-param mv_spacing];
        [/item-calc]
[/if-item-param]
[item-param child_fld]
</td>
<td>
        mv_level=[item-param mv_level],
        mv_increment=[item-param mv_increment],
        mv_children=[item-param mv_children]
</td>
</tr>
[/tree]
</table>
```

Accepts the following paremeters:

56.79.2.5. table

Database table which contains the tree. Must be a valid Interchange table identifier.

56.79.2.6. master

The column which is used to determine the parent of the item.

56.79.2.7. subordinate

The child column, which determines which items are sub-items of the current one. Used to re-query for items with its value in `master`.

56.79.2.8. start_item

The first item to be followed, i.e. the `master` value of all the top-level items.

56.79.2.9. autodetect

Specifies that the next level should be followed to detect the number of child items contained. Not recursive; only follows far enough to determine the children of the current item.

56.79.2.10. full

Specifies that all items should be followed. Essentially the same as specifying `memo` and passing the `explode` variable, but not dependent on them. Useful for building lists for inclusion in embedded Perl, among other things.

56.79.2.11. stop

An optional `stop` field which, when the value is true, can stop the following of the branch.

56.79.2.12. continue

An optional `continue` field which, when the value is true, can force the branch to be followed.

56.79.2.13. sort

The column which should be used for ordering the items −− determines the order in which they will be displayed under the current parent.

56.79.2.14. outline

Sets outline mode, where `mv_increment` will be displayed with letter values or numeral values. If set to specifically `1`, will produced outline increments like:

```
1
    A
    B
        1
        2
    C
        1
        2
            a
            b
                1
                2
                    a
                    b
2
```

56.79.2.15. memo

Indicates that the collapse/explode/toggle features are to be used, and names a `Scratch` variable where the values should be stored.

56.79.2.16. collapse

The name of a variable in the user's session which will determine that the tree should be "collapsed". When collapsed, the child items will not be followed unless they are set to be followed with `toggle`. Zeros all toggles.

Requires `memo` to be set if values are to be retained.

56.79.2.17. toggle

The name of a variable in the user's session which will determine that the current item should be either followed or not followed. The first time the `toggle` variable corresponding to its primary key is passed, the item will be expanded. The next call will "collapse" the item.

Requires `memo` to be set if values are to be retained.

56.79.2.18. explode

The name of a variable in the user's session which will determine that the tree should be "exploded". When exploded, all child items are followed and the full tree can be displayed.

Requires `memo` to be set if values are to be retained.

56.79.2.19. pedantic

When set to a true value, and an endless tree is detected (i.e. the child branch contains a parent) then the error will be logged to the catalog error log and the tree call will return with an error.

If `pedantic` is not set (the default), the current leaf will be shown but never followed. This allows partial display of the tree.

56.79.2.20. log_error

When set to a true value, and an endless tree is detected (i.e. the child branch contains a parent) then the error will be logged to the catalog error log. No logging done by default.

56.79.2.21. show_error

When set to a true value, and an endless tree is detected (i.e. the child branch contains a parent) then the error will be returned in the page. Errors are NOT shown by default.

In addition to the above values, all valid options for a list tag are in force. For example, you can set a "SELECTED" value on an option list with `option=1`, set the tag prefix with `prefix`, etc.

56.80. try

Allows you to trap errors. Interchange processes the body text of the [`try`][`/try`] block and returns it normally if it does not generate an error. If it does generate an error, interchange executes the [`catch`][`/catch`] block.

See also '`catch`'.

56.80.1. Summary

```
[try label=my_label other_named_attributes]
    Body text to return if no error
[/try]
.
.
.
[catch my_label]
    Body text to return if try block caused an error
[/catch]
```

Parameters	*Description*	*Default*
label	The label shared by the paired `try` and `catch` blocks	'default'

Attributes	*Description*	*Default*

status	Returns 0 (failed) or 1 (succeeded) instead of page output	*none*
hide	Suppresses page output	*No*
clean	Suppress `try` block output if it has an error	*No*
interpolate	See Interpolating Parameters	*No*
reparse	See Interpolating Parameters	*Yes*

Other_Charactreristics	
Invalidates cache	*No*
Container tag	*Yes*

Tag expansion example:

```
   [set divisor]0[/set]
   [try label=div]
      [calc] 1 / [scratch divisor] [/calc]
   [/try]
   [catch div]Division error[/catch]
---
   Division Error
```

ASP−like Perl call:

```
    $Tag->try(    { label  => I<'try_catch_label'>,
                    status => 1, },
                  $try_body  );

    $Tag->catch(  { label  => I<'try_catch_label'>, },
                  $catch_body )
```

or similarly with positional parameters,

```
    $Tag->try($label, $attribute_hash_reference, $try_body);
    $Tag->catch($label, $attribute_hash_reference, $catch_body);
```

56.80.1.1. See Also

`catch`

56.80.2. Description

Allows you to trap errors. Interchange processes the body text of the [`try`][`/try`] block and returns it normally if it does not generate an error. If it does generate an error, interchange executes the [`catch`][`/catch`] block. The catch block executes where it is on the page (*i.e.*, it does not replace the output of the try block).

Note that the `catch` block must occur after the [try] block in the document.

56.80.2.1. label

The try and catch blocks are matched by this label.

Technical note:

The try tag will also place a result in the $Session object. For example, the following returns the 'Illegal division by zero...' error message if it occurs:

```
[try label=divide][calc] 1 / [scratch divisor] [/calc][/try]

[catch divide]
   [calc]$Session->{try}{divide}[/calc]
[/catch]
```

The $Session->{try}{divide} object will be set to the empty string ('') if there was no error, or it will contain the error message if there was an error.

The [perl] and [calc] tags also set $Session->{try}->{*active_label*} on errors.

56.80.2.2. status

Suppresses `try` block output and returns 1 if no error or 0 if an error occurred instead. Executes the `catch` block as usual in case of an error.

56.80.2.3. hide

Suppresses `try` block output (regardless of success or failure). Executes the `catch` block as usual in case of an error.

56.80.2.4. clean

Setting '`clean=1`' will cause the `try` block to suppress its output only if it has an error. Otherwise (`clean=0` or not set), the `try` block will return whatever partial output it has completed before the error. The catch block will work as usual.

56.81. update

Forces an update of the specified interchange function. Function may be one of the following:

- **cart** (updates current or named cart)
- **process** (updates order or search)
- **values** (updates user-entered fields)
- **data** (updates database, using current **mv_** CGI form variables)

56.81.1. Summary

Parameters: **function**

- function
  - ♦ cart
    - ◊ Upates current or named cart (see `name` attribute)
  - ♦ process
    - ◊ Updates an order or a search page
  - ♦ values
    - ◊ Updates user-entered fields
  - ♦ data

◊ Updates database, using current **mv_** CGI form variables, for example:
· **mv_data_table** Table to update
· **mv_data_key** Key into table
· **mv_data_fields** Fields to update (space or null delimited)
· **mv_data_function** One of the following:
- delete
- update
- insert
- delete

· etc.

- name
  - ♦ Cart name to update (if 'function=cart')
  - ♦ Default: current cart

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->update(
        {
         function => VALUE,
        }
    )

OR

    $Tag->update($function, $ATTRHASH);
```

56.81.2. Description

Forces an update of the specified interchange function. Function may be one of the following:

- **cart** (updates current or named cart)
- **process** (updates order or search)
- **values** (updates user−entered fields)
- **data** (updates database, using current **mv_** CGI form variables)

56.82. userdb

56.82.1. Summary

Parameters: **function**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag−>userdb(
        {
         function => VALUE,
        }
    )

 OR

    $Tag−>userdb($function, $ATTRHASH);
```

Attribute aliases

```
            name ==> nickname
            table ==> db
```

56.82.2. Description

Interchange provides a `[userdb ...]` tag to access the UserDB functions.

```
 [userdb
        function=function_name
        username="username"*
        password="password"*
        verify="password"*
        oldpass="old password"*
        shipping="fields for shipping save"
        billing="fields for billing save"
        preferences="fields for preferences save"
        force_lower=1
        param1=value*
        param2=value*
        ...
        ]
```

* Optional

It is normally called in an `mv_click` or `mv_check` setting, as in:

```
    [set Login]
    mv_todo=return
    mv_nextpage=welcome
    [userdb function=login]
    [/set]

    <FORM ACTION="[process−target]" METHOD=POST>
    <INPUT TYPE=hidden NAME=mv_click VALUE=Login>
    Username <INPUT NAME=mv_username SIZE=10>
    Password <INPUT NAME=mv_password SIZE=10>
    </FORM>
```

There are several global parameters that apply to any use of the `userdb` functions. Most importantly, by default the database table is set to be *userdb*. If you must use another table name, then you should include a `database=table` parameter with any call to `userdb`. The global parameters (default in parens):

```
database     Sets user database table (userdb)
show         Show the return value of certain functions
             or the error message, if any (0)
force_lower  Force possibly upper-case database fields
             to lower case session variable names (0)
billing      Set the billing fields (see Accounts)
shipping     Set the shipping fields (see Address Book)
preferences  Set the preferences fields (see Preferences)
bill_field   Set field name for accounts (accounts)
addr_field   Set field name for address book (address_book)
pref_field   Set field name for preferences (preferences)
cart_field   Set field name for cart storage (carts)
pass_field   Set field name for password (password)
time_field   Set field for storing last login time (time)
expire_field Set field for expiration date (expire_date)
acl          Set field for simple access control storage (acl)
file_acl     Set field for file access control storage (file_acl)
db_acl       Set field for database access control storage (db_acl)
```

56.83. value

Returns the the current value of the named form input field. HTML−escapes Interchange tags in the result for security.

Can also set a new value within the current page.

56.83.1. Summary

```
[value name]
[value name=form_var_name other_named_attributes]
```

Parameters	*Description*	*Default*
name	This is the name of the form variable whose value you want.	*None*

Attributes	*Default*
set	*none*
hide	*No*
filter	*none*
keep (with filter)	*No*
scratch	*No*
default	*none*
interpolate (reparse)	*No*

Other_Charactreristics	
Invalidates cache	*Yes*

Tag expansion example:

Assuming form variable 'foo' = 'bar',

```
   [value foo]
---
   bar
```

ASP−like Perl call:

```
    $Tag->value(  { name => var_name, }  );

    # or if you simply want the value,
    $::Values->{var_name};
```

or similarly with positional parameters,

```
    $Tag->value($name$, $attribute_hash_reference);
```

56.83.2. Description

HTML examples:

```
   <PARAM MV="value name">
   <INPUT TYPE="text" NAME="name" VALUE="[value name]">
```

Expands into the current value of the named customer/form input field. When the value is returned, any Interchange tags present in the value will be escaped. This prevents users from entering Interchange tags in form values, which would be a serious security risk.

56.83.2.1. name

This is the name of the form variable whose value you want.

56.83.2.2. set

You can change a value with '`set`=*new_value*'. The tag will return the value you set unless you also set the hide=1 attribute.

Use this to "uncheck" a checkbox or set other form variable values to defaults. If you simply want a place to store your own data, use the `set` and `scratch` tags instead.

Note that this is only available in new−style tags, for safety reasons.

56.83.2.3. hide

Setting `hide`=1 suppresses the tag's return value, which can be useful with the `set`set attribute.

56.83.2.4. filter

See the `filter` tag for a list of filters.

Setting '`filter`="*filter*"' modifies the named value with the specified filter.

56.83.2.5. keep (with filter)

Set `keep`=1 if you want the tag to return a filtered result but do not want the filter to modify the form value itself in the $::Values hash.

56.83.2.6. scratch

Setting '`scratch`=1' places a copy of the value in the $::Scratch hash. An illustrative example:

```
foo is [value name=foo scratch=1] in the Values hash
foo is now also [scratch foo] in the Scratch hash
```

56.83.2.7. default

This sets a return value in case the named value is missing or otherwise false. The following will expand to "Using default":

```
[value name=myname set=0 hide=1]
[value myname default="Using default"]
```

56.84. value_extended

56.84.1. Summary

Parameters: **name**

Positional parameters in same order.

The attribute hash reference is passed to the subroutine after the parameters as the last argument. **This may mean that there are parameters not shown here.**

Must pass named parameter interpolate=1 to cause interpolation.

Invalidates cache: **YES**

Called Routine:

ASP−like Perl call:

```
    $Tag->value_extended(
        {
         name => VALUE,
        }
    )

OR

    $Tag->value_extended($name, $ATTRHASH);
```

56.84.2. Description

Named call example:

```
[value-extended
         name=formfield
         outfile=filename*
         ascii=1*
         yes="Yes"*
         no="No"*
         joiner="char|string"*
         test="isfile|length|defined"*
         index="N|N..N|*"
         file_contents=1*
         elements=1*]
```

Expands into the current value of the customer/form input field named by field. If there are multiple elements of that variable, it will return the value at `index`; by default all joined together with a space.

If the variable is a file variable coming from a multipart/form-data file upload, then the contents of that upload can be returned to the page or optionally written to the `outfile`.

56.84.2.1. name

The form variable NAME. If no other parameters are present, then the value of the variable will be returned. If there are multiple elements, then by default they will all be returned joined by a space. If `joiner` is present, then they will be joined by its value.

In the special case of a file upload, the value returned is the name of the file as passed for upload.

56.84.2.2. joiner

The character or string that will join the elements of the array. Will accept string literals such as "\n" or "\r".

56.84.2.3. test

Three tests -- `isfile` returns true if the variable is a file upload. `length` returns the length. `defined` returns whether the value has ever been set at all on a form.

56.84.2.4. index

The index of the element to return if not all are wanted. This is useful especially for pre-setting multiple search variables. If set to `*`, will return all (joined by `joiner`). If a range, such as `0 .. 2`, will return multiple elements.

56.84.2.5. file_contents

Returns the contents of a file upload if set to a non-blank, non-zero value. If the variable is not a file, returns nothing.

56.84.2.6. outfile

Names a file to write the contents of a file upload to. It will not accept an absolute file name; the name must be relative to the catalog directory. If you wish to write images or other files that would go to HTML space, you must use the HTTP server's `Alias` facilities or make a symbolic link.

56.84.2.7. ascii

To do an auto−ASCII translation before writing the `outfile`, set the `ascii` parameter to a non−blank, non−zero value. Default is no translation.

56.84.2.8. yes

The value that will be returned if a test is true or a file is written successfully. Defaults to `1` for tests and the empty string for uploads.

56.84.2.9. no

The value that will be returned if a test is false or a file write fails. Defaults to the empty string.

57. User−defined Tags

To define a tag that is catalog−specific, place *UserTag* directives in your catalog.cfg file. For server−wide tags, define them in interchange.cfg. Catalog−specific tags take precedence if both are defined −− in fact, you can override the base Interchange tag set with them. The directive takes the form:

```
UserTag  tagname  property  value
```

where `tagname` is the name of the tag, `property` is the attribute (described below), and `value` is the value of the property for that tagname.

The user tags can either be based on Perl subroutines or just be aliases for existing tags. Some quick examples are below.

An alias:

```
UserTag product_name Alias     data products title
```

This will change [product_name 99−102] into [data products title 99−102], which will output the `title` database field for product code `99−102`. Don't use this with `[item−data ...]` and `[item−field ...]`, as they are parsed separately. You can do `[product−name [item−code]]`, though.

A simple subroutine:

```
UserTag company_name Routine   sub { "Your company name" }
```

When you place a [company−name] tag in an Interchange page, the text `Your company name` will be substituted.

A subroutine with a passed text as an argument:

```
UserTag caps   Routine   sub { return "\U@_" }
UserTag caps   HasEndTag
```

The tag [caps]This text should be all upper case[/caps] will become `THIS TEXT SHOULD BE ALL UPPER CASE`.

Here is a useful one you might wish to use:

```
UserTag quick_table HasEndTag
UserTag quick_table Interpolate
UserTag quick_table Order   border
UserTag quick_table Routine <<EOF
sub {
    my ($border,$input) = @_;
    $border = " BORDER=$border" if $border;
    my $out = "<TABLE ALIGN=LEFT$border>";
    my @rows = split /\n+/, $input;
    my ($left, $right);
    for(@rows) {
        $out .= '<TR><TD ALIGN=RIGHT VALIGN=TOP>';
        ($left, $right) = split /\s*:\s*/, $_, 2;
        $out .= '<B>' unless $left =~ /</;
        $out .= $left;
```

```
            $out .= '</B>' unless $left =~ /</;
            $out .= '</TD><TD VALIGN=TOP>';
            $out .= $right;
            $out .= '</TD></TR>';
            $out .= "\n";
        }
        $out .= '</TABLE>';
    }
    EOF
```

Called with:

```
    [quick-table border=2]
    Name: [value name]
    City: [value city][if value state], [value state][/if] [value country]
    [/quick_table]
```

As is the case with [perl] tag, user tags run under the Perl Safe.pm module with warnings disabled. Unlike [perl] tags, however, user tags use Perl's 'strict' pragma.

The properties for UserTag are:

57.1. Alias

An alias for an existing (or other user-defined) tag. It takes the form:

```
    UserTag tagname Alias    tag to insert
```

An Alias is the only property that does not require a *Routine* to process the tag.

57.2. attrAlias

An alias for an existing attribute for defined tag. It takes the form:

```
    UserTag tagname attrAlias   alias attr
```

As an example, the standard Interchange `value` tag takes a named attribute of `name` for the variable name, meaning that [value name=var] will display the value of form field `var`. If you put this line in catalog.cfg:

```
    UserTag value attrAlias   identifier name
```

then [value identifier=var] will be an equivalent tag.

57.3. CanNest

Notifies Interchange that this tag must be checked for nesting. Only applies to tags that have *HasEndTag* defined, of course. NOTE: Your routine must handle the subtleties of nesting, so don't use this unless you are quite conversant with parsing routines. See the routines `tag_loop_list` and `tag_if` in lib/Vend/Interpolate.pm for an example of a nesting tag.

```
    UserTag tagname CanNest
```

57.4. HasEndTag

Defines an ending [/tag] to encapsulate your text −− the text in between the beginning `[tagname]` and ending `[/tagname]` will be the last argument sent to the defined subroutine.

```
UserTag tagname HasEndTag
```

57.5. Implicit

This defines a tag as implicit, meaning it can just be an `attribute` instead of an `attribute=value` pair. It must be a recognized attribute in the tag definition, or there will be big problems. Use this with caution!

```
UserTag tagname Implicit attribute value
```

If you want to set a standard include file to a fixed value by default, but don't want to have to specify [`include` file="/long/path/to/file"] every time, you can just put:

```
UserTag include Implicit file file=/long/path/to/file
```

and [`include` file] will be the equivalent. You can still specify another value with [`include` file="/another/path/to/file"].

57.6. InsertHTML

This attribute makes HTML tag output be inserted into the containing tag, in effect adding an attribute=value pair (or pairs).

```
UserTag tagname InsertHTML   htmltag  mvtag|mvtag2|mvtagN
```

In Interchange's standard tags, among others, the <OPTION ...> tag has the [selected ..] and [checked ...] tags included with them, so that you can do:

```
<INPUT TYPE=checkbox
     MV="checked mvshipmode upsg" NAME=mv_shipmode> UPS Ground shipping
```

to expand to this:

```
<INPUT TYPE=checkbox CHECKED NAME=mv_shipmode> UPS Ground shipping
```

Providing, of course, that `mv_shipmode` **is** equal to `upsg`. If you want to turn off this behavior on a per−tag basis, add the attribute mv.noinsert=1 to the tag on your page.

57.7. InsideHTML

To make a container tag be placed **after** the containing HTML tag, use the InsideHTML setting.

```
UserTag tagname InsideHTML   htmltag  mvtag|mvtag2|mvtagN
```

In Interchange's standard tags, the only InsideHTML tag is the <SELECT> tag when used with *loop*, which causes this:

```
<SELECT MV="loop upsg upsb upsr" NAME=mv_shipmode>
<OPTION VALUE="[loop-code]"> [shipping-desc [loop-code]]
</SELECT>
```

to expand to this:

```
<SELECT NAME=mv_shipmode>
[loop upsg upsb upsr]
<OPTION VALUE="[loop-code]"> [shipping-desc [loop-code]]
[/loop]
</SELECT>
```

Without the InsideHTML setting, the [loop ...] would have been **outside** of the select -- not what you want. If you want to turn off this behavior on a per-tag basis, add the attribute mv.noinside=1 to the tag on your page.

57.8. Interpolate

The behavior for this attribute depends on whether the tag is a container (i.e. `HasEndTag` is defined). If it is not a container, the `Interpolate` attribute causes the **the resulting HTML** from the `UserTag` to be re-parsed for more Interchange tags. If it is a container, `Interpolate` causes the contents of the tag to be parsed **before** the tag routine is run.

```
UserTag tagname Interpolate
```

57.9. InvalidateCache

If this is defined, the presence of the tag on a page will prevent search cache, page cache, and static builds from operating on the page.

```
UserTag tagname InvalidateCache
```

It does not override [tag flag build][/tag], though.

57.10. Order

The optional arguments that can be sent to the tag. This defines not only the order in which they will be passed to *Routine*, but the name of the tags. If encapsulated text is appropriate (*HasEndTag* is set), it will be the last argument.

```
UserTag tagname Order param1 param2
```

57.11. PosRoutine

Identical to the Routine argument -- a subroutine that will be called when the new syntax is not used for the call, i.e. `[usertag argument]` instead of `[usertag ARG=argument]`. If not defined, *Routine* is used, and Interchange will usually do the right thing.

57.12. ReplaceAttr

Works in concert with InsertHTML, defining a **single** attribute which will be replaced in the insertion operation..

```
UserTag tagname ReplaceAttr  htmltag attr
```

An example is the standard HTML <A HREF=...> tag. If you want to use the Interchange tag [area pagename] inside of it, then you would normally want to replace the HREF attribute. So the equivalent to the following is defined within Interchange:

```
UserTag  area  ReplaceAttr  a  href
```

Causing this

```
<A MV="area pagename" HREF="a_test_page.html">
```

to become

```
<A HREF="http://yourserver/cgi/simple/pagename?X8sl2lly;;44">
```

when intepreted.

57.13. ReplaceHTML

For HTML−style tag use only. Causes the tag containing the Interchange tag to be stripped and the result of the tag to be inserted, for certain tags. For example:

```
UserTag company_name Routine sub { my $l = shift; return "$l: XYZ Company" }
UserTag company_name HasEndTag
UserTag company_name ReplaceHTML  b    company_name
```


 is the HTML tag, and "company_name" is the Interchange tag. At that point, the usage:

```
<B MV="company−name"> Company </B>  −−−>>  Company: XYZ Company
```

Tags not in the list will not be stripped:

```
<I MV="company−name"> Company </I> −−−>>  <I>Company: XYZ Company</I>
```

57.14. Routine

An inline subroutine that will be used to process the arguments of the tag. It must not be named, and will be allowed to access unsafe elements only if the `interchange.cfg` parameter *AllowGlobal* is set for the catalog.

```
UserTag tagname Routine  sub { "your perl code here!" }
```

The routine may use a "here" document for readability:

```
UserTag tagname Routine <<EOF
```

```
sub {
    my ($param1, $param2, $text) = @_;
    return "Parameter 1 is $param1, Parameter 2 is $param2";
}
EOF
```

The usual *here documents* caveats apply.

Parameters defined with the *Order* property will be sent to the routine first, followed by any encapsulated text (*HasEndTag* is set).

Note that the UserTag facility, combined with AllowGlobal, allows the user to define tags just as powerful as the standard Interchange tags. This is not recommended for the novice, though −− keep it simple. 8−)

58. Standard Usertags

The distribution includes a number of prebuilt usertags in the `usertag` directory in the Interchange software directory. Some of these are used by the foundation catalog or its administrative interface.

58.1. bar_button

Displays content based on current page. Could be used for building e.g. menu bars.

58.1.1. Summary

```
[bar-button page current]...[selected]...[/selected][/bar-button]
[bar-button page=page current=current-page]...[selected]...[/selected][/bar-button]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	***Description***	***Default***
page	Name of page for which this bar-button is defined	*none*
current	Name of the current page	Current page: MV_PAGE

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	[/bar-button]

Tag expansion example:

To build a simple '3-button' menu bar one could put the following on each of the pages. The results of this code for page2 are shown below.

```
        <table><tr>
        [bar-button page=page1]
        <TD><A HREF="[area page1]">PAGE-1</A></TD>
        [selected]
        <TD bgcolor="red"><A HREF="[area page1]"><B>PAGE-1-selected</B></A></TD>
        [/selected]
        [/bar-button]
        [bar-button page=page2]
        <TD><A HREF="[area page2]">PAGE-2</A></TD>
        [selected]
        <TD bgcolor="red"><A HREF="[area page2]"><B>PAGE-2-selected</B></A></TD>
        [/selected]
        [/bar-button]
        [bar-button page=page3]
        <TD><A HREF="[area page3]">PAGE-3</A></TD>
        [selected]
        <TD bgcolor="red"><A HREF="[area page3]"><B>PAGE-3-selected</B></A></TD>
        [/selected]
        [/bar-button]
        </tr></table>
--------------------------------------------------
        PAGE-1    PAGE-2-selected    PAGE-3
```

ASP−like Perl call:

```
$Tag->button_bar(  { page => $page,
                  current  => $current,
                      body => $body}  );
```

or similarly,

```
$Tag->area($page, $current, $body);
```

58.1.1.1. See Also

bar_link routine

58.1.2. Description

Displays content based on current page. The content between the [selected][/selected] tags will be displayed only if the name of the curent page matches the name that was passed to the page parameter (page=page−name). The default content will be displayed when there is no match.

58.1.2.1. page

The name of the page for which this definition of the bar−button is defined.

58.1.2.2. current

The name of the current page. Defaults to current page MV_PAGE.

58.2. button

58.3. convert_date

This tag converts a given date format into another format.

58.3.1. Summary

```
[convert_date day* other_named_attributes>[/convert_date]
[convert_date day=n* other_named_attributes[/convert_date]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	***Description***	***Default***
days	The number of days from or before today	*none*

Attributes	***Default***
format	'%d−%b−%Y %I:%M%p' or '%d−%b−%Y'
fmt − Alias for format	*none*
raw	*none*

zerofix	*none*
Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	[/convert_date]

Tag expansion example:

```
  a. [convert-date][/convert-date]
  b. [convert-date 1][/convert-date]
  c. [convert-date -1][/convert-date]
  d. [convert-date]2001-5-1[/convert-date]
  e. [convert-date]2001-05-01[/convert-date]
  f. [convert-date]20010515[/convert-date]
  g. [convert-date raw=1]2001-05-18[/convert-date]
  h. [convert-date fmt="%d-%m-%Y"]2001-05-18[/convert-date]
  i. [convert-date]200 1 -  --051 =9[/convert-date]
  j. [convert-date]2001 -  --05 -20 11 1 5[/convert-date]
  k. [convert-date raw=1]2001-05-21 11:15[/convert-date]
--------------------------------------------------------
  a. 18-May-2001 03:15AM (todays day and time)
  b. 19-May-2001 03:15AM (today + 1 day)
  c. 17-May-2001 03:15AM (today - 1 day)
  d. 01-May-2001
  e. 01-May-2001
  f. 15-May-2001
  g. 20010518
  h. 18-05-2001
  i. 19-May-2001
  j. 20-May-2001 11:15AM
  k. 200105211115
```

ASP-like Perl call:

```
    $Tag->convert_date( { day => 1 } );

    $Tag->convert_date( { body => "2001-05-19 15:35",
                          format => "%d-%m-%Y %H:%M", } );
```

or similarly with positional parameters,

```
    $Tag->convert_date( 1 );
```

58.3.2. Description

This tag converts almost any given date format into another, possibly user defined, format.

58.3.2.1. days

Number of days from or before today's date and time. Will only be used if nothing is supplied between the tags.

58.3.2.2. format

POSIX time format string of your choice. See Unix strftime(3) manpage for complete details.

58.3.2.3. raw

If this option is set to true, will display given date in raw format, e.g. yyyymmdd or yyyymmddHHMM.

58.3.2.4. zerofix

Strips leading zeroes from numbers.

58.4. db_date

This tag returns the time of last access of the database source file.

58.4.1. Summary

```
[db_date table* format*]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	***Description***	***Default***
table	Table name.	products
format	POSIX time format string	%A %d %b %Y

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	*No*

Tag expansion example:

```
   [db-date]
   [db-date cat]
   [db-date table=cat format="%d %b %Y"]
-----------------------------------------------------
   Wednesday 02 May 2001 (products.txt)
   Wednesday 03 May 2001 (cat.txt)
   03 May 2001 (cat.txt)
```

ASP-like Perl call:

```
    $Tag->db_date( { table => cat,
                     format => "%d %b %Y", } );
```

or similarly with positional parameters,

```
    $Tag->db_date( "cat", "%d %b %Y" );
```

58.4.2. Description

This tag returns the time of last access of the database source file.

58.4.2.1. table

Table name. Defaults to products if not specified.

58.4.2.2. format

POSIX time format string. See Unix strftime(3) manpage for details. Defaults to '%A %d %b %Y' when not specified.

58.5. delete_cart

This tag deletes a cart from the userdb.

58.5.1. Summary

```
[delete_cart nickname]
[delete_cart nickname="cart-name"]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	***Description***	***Default***
nickname	Must be an existing nickname	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	*No*

Tag expansion example:

```
   [delete_cart mycart]
   [delete_cart nickname="mycart"]
-----------------------------------------------------
```

ASP-like Perl call:

```
$Tag->delete_cart( { nickname => "mycart", } );
```

or similarly with positional parameters,

```
$Tag->delete_cart( "mycart" );
```

58.5.1.1. See Also

userdb, load_cart, save_cart and pages templates/components/saved_carts_list_small, pages/saved_carts.html for more examples.

58.5.2. Description

Deletes a cart with name nickname from the user database. Basically the same as [userdb function=delete_cart nickname=mcart].

58.5.2.1. nickname

Nickname of cart to be deleted.

58.6. email

This tag takes a recipient address and a body text and uses the SendmailProgram with −t option to send email.

58.6.1. Summary

```
[email to subject* reply* from* extra*]Your message[/email]
[email to=to_address subject=subject reply=reply_address
    from=from_address extra=extra_headers]Your message[/email]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	*Description*	*Default*
to	Email address of recipient	*none*
subject	Subject line	String: <no subject>
reply	Email address to be used for the reply−to header	*none*
from	Senders email address	First address in MailOrderTo configuration variable
extra	Additional headers to be included	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	[/email]

Tag expansion example:

```
[email
       to="foo@bar.com"
  subject="Greetings"
     from="bar@foo.com"
]
Hello World
[/email]
```

ASP−like Perl call:

```
$Tag->email(  { to => $to,
```

```
                from => $from,
             subject => $subject,
               reply => $reply,
               extra => $extra,
                body => $body }  );
```

or similarly,

```
$Tag->email($to, $subject, $reply, $from, $extra, $body);
```

58.6.1.1. See Also

email_raw and etc/mail_receipt, pages/process_return.html, pages/stock−alert−added.html for examples.

58.6.2. Description

Will send the content between the [email][/email] tags as an email to the recipient (to) using the SendmailProgram with −t option.

58.6.2.1. extra

Extra headers to be included. Example: Errors−To: errors@yourdomain.com

58.6.2.2. from

Email address identifying the sender of the message. Will use the first email address of the MailOrderTo configuration variable if it is not supplied.

58.6.2.3. reply

Email address to be used for the Reply−to header. No Reply−to header will be present if this parameter is omitted.

58.6.2.4. subject

Short text describing the content of the message. The Subject line of an email message. The string <no subject> will be substituted if this parameter is omitted.

58.6.2.5. to

Valid email address(es) of the recipient(s). This parameter is required.

58.7. email_raw

This tag takes a raw email message, **including headers**, and uses the SendmailProgram with −t option.

58.7.1. Summary

```
[email_raw]Your message incuding headers[/email_raw]
```

Other_Charactreristics	

Invalidates cache	*No*
Macro	*No*
Has end tag	[/email_raw]

Tag expansion example:

```
   [email_raw]
From: foo@bar.com
To: bar@foo.com
Subject: baz

The text of the message.
   [/email_raw]
----------------------------------------------------
```

The headers must be at the beginning of the line, and the header must have a valid To: or it will not be delivered.

ASP−like Perl call:

```
    $Tag->email_raw(  { body => $body }  );
```

or similarly,

```
    $Tag->email_raw($body);
```

58.7.1.1. See Also

email

58.7.2. Description

Will send the content between the [email_raw][/email_raw] tags as a raw email message to the recipient specified in the supplied headers using the SendmailProgram with −t option.

58.8. fcounter

58.9. fedex_query

58.10. formel

58.11. get−url

Fetch a URL and return the contents.

58.11.1. Summary

```
    [get-url url]
    [get-url url="valid-url" strip=1*]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	*Description*	*Default*
url	Must be a valid URL. Meaning, you have to supply the protocol. Example ♦ http://demo.akopia.com/ ♦ ftp://ftp.akopia.com/	*none*

Attributes	*Default*
strip	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	*No*

Tag expansion example:

```
   [get-url http://demo.akopia.com/]
   [get-url url="http://demo.akopia.com/" strip=1]
-----------------------------------------------------
```

ASP-like Perl call:

```
    $Tag->get_url( { url => "http://demo.akopia.com/", } );

    $Tag->get_url( { url => "http://demo.akopia.com/",
                     strip => 1, } );
```

or similarly with positional parameters,

```
    $Tag->get_url( "http://demo.akopia.com/" );
```

58.11.2. Description

Uses the LWP libraries (LWP::Simple) to fetch a URL and returns the contents.

58.11.2.1. strip

If the strip option is set, strips everything up to <body> and everything after </body>.

58.11.2.2. url

Must be a valid URL (including protocol).

58.12. load_cart

This tag loads a cart by name from the userdb.

58.12.1. Summary

```
[load_cart nickname]
[load_cart nickname="cart-name"]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	*Description*	*Default*
nickname	Must be an existing nickname. Nickname is constructed from: ♦ a name part ♦ time modified (time the cart was saved by save_cart tag) ♦ type (c for cart, r for recurring)	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	*No*

Tag expansion example:

```
   [load_cart mycart:990102732:c]
   [load_cart nickname="mycart:990102732:c"]
-----------------------------------------------------
```

ASP-like Perl call:

```
$Tag->load_cart( { nickname => "mycart:990102732:c", } );
```

or similarly with positional parameters,

```
$Tag->load_cart( "mycart:990102732:c" );
```

58.12.1.1. See Also

userdb, delete_cart, save_cart and pages templates/components/saved_carts_list_small, pages/saved_carts.html for more examples.

58.12.2. Description

Loads a cart with name nickname from the user database. It will be merged with the current cart. Basically the same as [userdb function=get_cart nickname=cartname merge=1].

58.12.2.1. nickname

Nickname of cart to be loaded. See above.

58.13. loc

58.14. rand

58.15. reconfig

58.16. reconfig_time

58.17. reconfig_wait

58.18. save_cart

This tag saves the current cart or recurring order in the userdb under a given name.

58.18.1. Summary

```
[save_cart nickname recurring]
[save_cart nickname="cart-name" recurring=1]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	***Description***	***Default***
nickname	Label for the cart.	*none*
recurring	Set to true if recurring. Set to false, or ommit if cart.	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	*No*

Tag expansion example:

```
   [save_cart mycart]
   [save_cart nickname=mycart recurring=1]
```

ASP-like Perl call:

```
$Tag->save_cart( { nickname => mycart,
                   recurring => 1, } );
```

or similarly with positional parameters,

```
$Tag->save_cart( "mycart", "1" );
```

58.18.1.1. See Also

userdb, delete_cart, load_cart and pages templates/components/saved_carts_list_small, pages/saved_carts.html for more examples.

58.18.2. Description

Saves the current cart with name nickname in the user database. Basically the same as [userdb function=set_cart nickname=cartname]

58.18.2.1. nickname

Nickname for the current cart to be saved. You can use same nickname for different carts. An index will be added if there are more carts with the same nickname.

58.18.2.2. recurring

Set to true if recurring. Set to false, or simply ommit it, if it is a cart.

58.19. summary

This tag calculates column totals.

58.19.1. Summary

```
[summary amount]
[summary amount=n.nn other_named_attributes]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	***Description***	***Default***
amount	Numerical value to be added to previous total	*none*

Attributes	***Default***
currency	*none*
format	*none*
hide	*none*, no hiding
name	ONLY0000, internal use only
reset	*none*
total	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	*No*

Tag expansion example:

```
   [loop list="10 20 30.5"]
   [summary amount="[loop-code]" hide=1]
   [/loop]
   [summary total=1 format="%.3f"]
   [summary total=1 currency=1]
-----------------------------------------------------
   60.500
   $60.50
```

ASP-like Perl call:

```
    $Tag->summary( { amount => 10.5,
                     hide => 1, } );

    $Tag->summary( { amount => 25,
                     name => mytotal,
                 currency => 1, } );
```

or similarly with positional parameters,

```
    $Tag->summary( 10.5, $attribute_hash_reference );
```

58.19.1.1. See Also

templates/components/cart, pages/ord/checkout.html for more examples.

58.19.2. Description

The summary tag provides you with an easy way to calculate and display totals. The display of the amounts is fully customizable. You can hide diplay, or you can show the amounts with the proper currency formatting according to the locale, or you can define your own formatting. Any number of summaries can be kept on a page.

58.19.2.1. currency

The amount or total will be diplayed according to the currency formatting of the current locale if this attribute is set to true (non blank or zero).

58.19.2.2. format

You can choose any formatting of the amount you like. Just set the format attribute to the desired formatting string (%s, %.2f etc.). When both, currency and format attributes are set, the format attribute will take precedence. So it doesn't make much sense to set them both at the same time.

58.19.2.3. hide

Will suppress the display of amount when set to true.

58.19.2.4. name

You can calculate as many totals as you like on the same page. Just supply a different label for each summary.

58.19.2.5. reset

Will erase the total(s) if set to true. Be careful tough. It will reset ALL totals when you have no name attribute supplied. If you have provided a label for the name atribute then it will only reset the total for that particular label. All others won't be touched.

58.19.2.6. total

Will show the total instead of the amount if set to true.

58.20. table_organize

Takes an unorganized set of table cells and organizes them into rows based on the number of columns.

58.20.1. Summary

```
[table-organize cols* other_named_attributes]
        [loop ....] <td> [loop-tags] </td> [/loop]
    [/table-organize]

[table-organize cols=n* other_named_attributes]
        [loop ....] <td> [loop-tags] </td> [/loop]
    [/table-organize]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	*Description*	*Default*
cols	Number of columns.	2
columns	Alias for cols.	2

Attributes	*Default*
caption	*none*
columnize	*none*
embed	*none*
filler	
limit	*none*
pretty	*none*
rows	*none*
table	*none*

td	*none*
tr	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	[/table−organize]

Tag expansion example:

This example produces a table that (1) alternates rows with background colors "#EEEEEE" and "#FFFFFF", and (2) aligns the columns right, center, left:

```
        [table−organize
            cols=3
            pretty=1
            tr.0='bgcolor="#EEEEEE"'
            tr.1='bgcolor="#FFFFFF"'
            td.0='align=right'
            td.1='align=center'
            td.2='align=left'
            ]
            [loop list="1 2 3 1a 2a 3a 1b"] <td> [loop−code] </td> [/loop]
        [/table−organize]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
        <tr bgcolor="#EEEEEE">
                <td align=right>1</td>
                <td align=center>2</td>
                <td align=left>3</td>
        </tr>
        <tr bgcolor="#FFFFFF">
                <td align=right>1a</td>
                <td align=center>2a</td>
                <td align=left>3a</td>
        </tr>
        <tr bgcolor="#EEEEEE">
                <td align=right>1b</td>
                <td align=center> </td>
                <td align=left> </td>
        </tr>
```

If the attribute columnize=1 is present, the result will look like:

```
            <tr bgcolor="#EEEEEE">
                    <td align=right>1</td>
                    <td align=center>1a</td>
                    <td align=left>1b</td>
            </tr>
            <tr bgcolor="#FFFFFF">
                    <td align=right>2</td>
                    <td align=center>2a</td>
                    <td align=left> </td>
            </tr>
            <tr bgcolor="#EEEEEE">
                    <td align=right>3</td>
                    <td align=center>3a</td>
                    <td align=left> </td>
            </tr>
```

See the source for more ideas on how to extend this tag.

ASP−like Perl call:

```
$Tag->table_organize( { cols => 3,
                        pretty => 1, }, $BODY );
```

or similarly with positional parameters:

```
$Tag->table_organize( $cols, $attribute_hash_reference, $BODY );
```

58.20.1.1. See Also

pages/flypage.html, pages/quantity.html, templates/components/best_horizontal, templates/components/cart, templates/components/cross_horizontal, templates/components/random, templates/components/random_vertical, templates/components/upsell

58.20.2. Description

Takes an unorganized set of table cells and organizes them into rows based on the number of columns; it will also break them into separate tables.

If the number of cells are not on an even modulus of the number of columns, then "filler" cells are pushed on.

58.20.2.1. cols (or columns)

Number of columns. This argument defaults to 2 if not present.

58.20.2.2. rows

Optional number of rows. Implies "table" parameter.

58.20.2.3. table

If present, will cause a surrounding <TABLE> </TABLE> pair with the attributes specified in this option.

58.20.2.4. caption

Table <CAPTION> container text, if any. Can be an array.

58.20.2.5. td

Attributes for table cells. Can be an array.

58.20.2.6. tr

Attributes for table rows. Can be an array.

58.20.2.7. columnize

Will display cells in (newspaper) column order, i.e. rotated.

58.20.2.8. pretty

Adds newline and tab characters to provide some reasonable indenting.

58.20.2.9. filler

Contents to place in empty cells put on as filler. Defaults to " ".

58.20.2.10. limit

Maximum number of cells to use. Truncates extra cells silently.

58.20.2.11. embed

If you want to embed other tables inside, make sure they are called with lower case <td> elements, then set the embed tag and make the cells you wish to organize be <TD> elements. To switch that sense, and make the upper−case or mixed case be the ignored cells, set the embed parameter to "lc".

```
[table−organize embed=lc]
            <td>
                    <TABLE>
                            <TR>
                            <TD> something
                            </TD>
                            </TR>
                    </TABLE>
            </td>
[/table−organize]
```

or

```
[table−organize embed=uc]
            <TD>
                    <table>
                            <tr>
                            <td> something
                            </td>
                            </tr>
                    </table>
            </TD>
[/table−organize]
```

The "tr", "td", and "caption" attributes can be specified with indexes; if they are, then they will alternate according to the modulus.

The "td" option array size should probably always equal the number of columns; if it is bigger, then trailing elements are ignored. If it is smaller, no attribute is used.

58.21. title_bar

Creates a quick title bar.

58.21.1. Summary

```
[title-bar width size color]My title[/title-bar]
[title-bar width=600 size=5 color="#ff0000"]My title[/title-bar]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	*Description*	*Default*
color	Background color the bar. Defaults to ♦ variable HEADERBG or ♦ #444444	HEADERBG or #444444
size	Font size	6
width	Width of the title bar	500

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	[/title−bar]

Tag expansion example:

```
   [title-bar 600 5 red]My title[/title-bar]
   [title-bar width=600 size=5 color="#ff0000"]My title[/title-bar]
-----------------------------------------------------
```

ASP−like Perl call:

```
    $Tag->title_bar( { body => "My Title", } );

    $Tag->title_bar( { width => 400,
                       color => "#0000ff",
                        body => "My title", } );
```

or similarly with positional parameters,

```
    $Tag->title_bar( 600, 5, "red", "My title" );
```

58.21.2. Description

Quickly adds a title bar to your pages without having to type the html each time. Background color, width of the bar and size of the text can be customized by setting the appropriate parameter. The text color defaults to variable HEADERTEXT or when its not present to white.

58.21.2.1. color

Sets the background color of the bar. You can set the color as 'red', '#ff0000', or 'bgcolor="#ff0000"'.

58.21.2.2. size

Determines the size of the text. Parameter shoud be set to a value accepted by the HTML <font> tag size attribute.

58.21.2.3. width

Sets the width of the bar.

58.22. ups_query

58.23. usertrack

58.24. var

58.25. xml_generator

This is a quick and dirty tag that generates XML tags based upon one of two types of data (delimited and session).

58.25.1. Summary

```
[xml-generator type* other_named_attributes][/xml-generator]
[xml-generator type=value* other_named_attributes][/xml-generator]
[xml-generator type=value* other_named_attributes][][/xml-generator]

*Optional
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	*Description*	*Default*
type	Data type. Delimited or session	delimited

Attributes	*Default*
attributes	*none*
dbdump	*none*
delimiter	\t

field_names	
separator	\n
toplevel_tag	'table' for delimited type and 'session' for other type
record_tag	record
field_tag	field
key_name	*none*
spacer	[\s,]+
no_second	*none*
skip_empty	*none*

Other_Charactreristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	[/xml−generator]

Tag expansion example:

```
[xml−generator
    type=delimited
    attributes="date"
    date="[tag time]%d−%b−%Y[/tag]"
    toplevel_tag=products]code  description     price
[query list=1 sql="select sku, description, price from products" prefix=xml][xml−code]  [xml−para
[/query][/xml−generator]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
<products date="18−May−2001">
        <record key="os28113">
                <code>os28113</code>
                <description>The Claw Hand Rake</description>
                <price>14.99</price>
        </record>
        <record key="os28006">
                <code>os28006</code>
                <description>Painters Brush Set</description>
                <price>29.99</price>
        </record>
        ...
</products>
```

ASP−like Perl call:

```
    $Tag−>xml_generator( {type => delimited,
                      toplevel_tag => apex,  }, $BODY );
```

or similarly with positional parameters,

```
    $Tag−>xml_generator( $type, $attribute_hash_reference, $BODY );
```

58.25.2. Description

58.25.2.1. type

delimited

Accepts a delimited and separated (default is TAB delimiter and newline separator) list of records such as that generated by an '[item−list]', '[sql]', or '[loop search=""]' ITL tag.

session

When the type is not delimited, it can contain any hash reference into the Interchange session. Examples are:

```
values       The form values
scratch      Scratch values
errors       Error values
other        Any other Session key, for example "source" for
             [data session source]
```

If the value is a hash, then it will be sent as an XML record with the top level equal to "session", and a second_level tag equal to the hash name, and keys as separate XML container tags. If the parameter "that is equal to the type" is given, only those fields will be shown. Otherwise the entire hash will be shown. For example, this tag:

```
[xml−generator type="values" values="fname lname"][/xml−generator]
```

will generate:

```
<session>
        <values>
                <fname>First</fname>
                <lname>Last</lname>
        </values>
</session>
```

if it is a scalar, then only the second level will be done:

```
[xml−generator type="cybercash_id"][/xml−generator]
```

will do the equivalent of:

```
<session>
        <cybercash_id>[data session cybercash_id]</cybercash_id>
</session>
```

So bringing it all together, the following:

```
[xml−generator type="values scratch source"
               values="fname lname"
               scratch="downloads"][/xml−generator]
```

will generate:

```
<session>
        <values>
                <fname>First</fname>
                <lname>Last</lname>
        </values>
        <scratch>
                <downloads>0</downloads>
        </scratch>
```

```
                <source>Partner1</source>
        </session>
```

58.25.2.2. no_second

Prevents the second−level tags from being generated. Extending the last example in the "session" type above, this

```
        [xml−generator  type="values scratch source"
                        no_second=1
                        values="fname lname"
                        scratch="downloads"][/xml−generator]
will generate:

        <session>
                <fname>First</fname>
                <lname>Last</lname>
                <downloads>0</downloads>
                <source>Partner1</source>
        </session>
```

58.25.2.3. attributes

The attributes (if any) to pass on to the top level tag. For instance,

```
        [xml−generator
              attributes="date"
              date="[tag time]%d−%b−%Y[/tag]"
              toplevel_tag=order
        ][/xml−generator]

will generate a toplevel tag pair of:

        <order date="18−Mar−2001">
        </order>
```

58.25.2.4. dbdump

Will dump all tables in the catalog when this attribute is set true. Used attributes are "toplevel_tag", "record_tag", "field_tag", and "skip_empty" or default values (resp. 'table', 'record', 'field').

```
Output format:
    <database name="catalogname">
        <toplevel_tag name="tablename1">
            <record_tag key="value of first field−1">
                <field_tag name="fieldname1">fieldvalue1</field_tag>
                <field_tag name="fieldname2">fieldvalue2</field_tag>
                        </record_tag>
            <record_tag key="value of first field−2">
                <field_tag name="fieldname1">fieldvalue1</field_tag>
                <field_tag name="fieldname2">fieldvalue2</field_tag>
                        </record_tag>
            </toplevel_tag>
        <toplevel_tag name="tablename2">
            <record_tag key="value of first field−1">
                <field_tag name="fieldname1">fieldvalue1</field_tag>
                <field_tag name="fieldname2">fieldvalue2</field_tag>
                        </record_tag>
```

```
        </toplevel_tag>
</database>
```

Important note: All tables are read into memory. So be warned, this could be a real memory hog.

Ton Verhagen's proposal:

1. Add option to select tables. E.g. dump_tables="products cat area" and/or
2. Add option to select an output file. E.g. dump_file="tabledump.XML". Send output to file line by line.

58.25.2.5. delimiter

Character used as delimiter of fields in delimited data type. Defaults to a tab character.

58.25.2.6. field_names

Space or comma−delimited list of field names to be used for delimited data type. Should be in the same order as in the data list provided (between the tags).

Another way of providing the field names would be:

```
[xml−generator .....]fieldname−1    fieldname−2     fieldname−3
    [field value list
     delimited by option delimiter and
     separated by option separator][/xml−generator]
```

Note: Field name list **must** be tab delimited.

Ton Verhagen's humble opinion: This should change in future versions! Use option delimiter instead.

58.25.2.7. separator

Character used as line separator in list between [xml−separator][xml−separator] tags and in output 'session' data type. Defaults to a newline, "\n".

58.25.2.8. toplevel_tag

The toplevel tag name to use. Defaults to "table" for the 'dbdump mode' and delimited type, and "session" for the other.

58.25.2.9. record_tag

Defines the tag name for the record tag. Defaults to 'record'. Used for 'dbdump mode' and delimited type.

58.25.2.10. field_tag

Defines the tag name for the field tag. Defaults to 'field'. Only used in 'dbdump mode'.

58.25.2.11. key_name

Only used in delimited data type. Defines fieldname to determine key value in "record_tag".

```
<record_tag key="value of field with name defined by key_name"> ....
```

58.25.2.12. spacer

Character used as delimiter in type parameter definition and corresponding attributes. Defaults to '[\s,]+' (one or more whitespace or comma).

```
[xml-generator type="values|scratch"
                   values="value1|value2"
                              scratch="scratch1|scratch2"
                              spacer="|"
                              ][/xml-generator]
```

58.25.2.13. skip_empty

Only used in dbdump mode (dbdump=1). Will skip empty fields if this attribute is set true.

C. Tag Entry Format

58.26. dummy

Alias: **pedagogy**, **dummy_alias**

Note: [**dummy** ...], [**pedagogy** ...] and [**dummy−alias** ...] are equivalent. If the tag has an endtag, you must not mix aliases between the tag and its endtag (i.e., [dummy] ... [/pedagogy] would not work).

A short description of the tag goes here. This example (**dummy**) is not an actual tag.

C..1. Summary

```
[dummy first second][/dummy]
[dummy first=first_args second=second_args other_named_attributes][/dummy]
```

Positional parameters: The first line shows the usage with positional parameters (given in order). The second line shows usage with named parameters.

Parameters	***Description***	***Default***
first	The first positional parameter *Special arguments* ♦ 'special_value' −− any special arguments to the parameter that cause the tag to behave differently are listed here and described in detail in the **Description** section below. ♦ 'pig_latin' −− For this dummy tag, let's suppose that an argument of 'pig_latin' rewrites the body text in pig latin.	default value if the parameter is not given
second	Another example parameter	*none*
alias1	alias for **first** −− some parameters have aliases −− [dummy alias1="X"] is equivalent to [dummy first="X"]	Same default as **first**, of course

Attributes	***Default***
more	*none*
still−more	*none* (requires **more**)
other	*none*
interpolate (body)	*Yes*
reparse	*No*

Other_Characteristics	
Invalidates cache	*No*
Macro	*No*
Has end tag	[/dummy]

Tag expansion example:

```
   [dummy first=pig_latin second="Capitalize"]\
   Body text acted on by the tag goes here.[/dummy]
------------------------------------------------------
   OdyBay ExtTaY ActedAy OnAy YBay EThay AgTay OesGay \
   EreHay.
```

Reading the tag expansion example:

format: The tag is listed first. A blank line separates it from the expanded return value.
long lines: When this document must break a line from an example because it is too wide for the page, a trailing backslash indicates the continuation. Note that such a trailing backslash is *not* part of the actual tag syntax or expansion.

ASP−like Perl call:

```
    $Tag->dummy(  { first  => 'goober',
                    second => 'foobar',
                    more   => $snafu, },  $body_text  );
```

or similarly,

```
    $Tag->dummy($first, $second, $attribute_hash_reference, $body_text);
```

C..2. Description

More detailed tag description

C..2.1. first

Section describing the first parameter

C..2.1.1. special_value

Description of treatment of the special argument. For example, giving the page tag an **href** of 'scan' causes it to link to a search specification rather than a page.

C..2.2. more

`more` is a named attribute that applies to this tag.

C..2.3. still−more

`still−more` is a named attribute that applies to this tag only when the `more` attribute has been given. It would be an error to use the `still−more` attribute without specifying a value for `more`

C..2.4. other

'other' is another named attribute that applies to this tag.

D. Template Parsing Order

D.1. Standard Parsing

Under normal circumstances, the template page containing tags and HTML is parsed in the following order:

1. Any content in MV_AUTOLOAD is prepended to the template page.

2. Any [pragma] tags anywhere in the text are processed, and the specified pragmas are set.
   - Since [pragma] tags are preprocessed before any other content, reparse will not catch them, nor will they work if included in variables. Also, the specified pragma will apply to the entire template (not just the section after the tag).
   - If you want to apply a pragma with a variable or only to part of a document, you must use [tag pragma="..."] instead.

3. Variables (macros) are processed in the following order:
   1. @@VARNAME@@ global variables
   2. @_VARNAME_@ local or global variables
   3. __VARNAME__ local variables

4. Interchange comments are stripped.

5. False−endtag macros are expanded (e.g., [/page] and [/order]).

6. '<!−−[tagname]−−>' escapes are converted to [tagname]
   - This can be a convenience for your HTML editor if it has trouble with tags using the standard [tagname] syntax.
   - However, if you want to HTML−comment out an Interchange tag in content that will be fed raw to a browser, you must include whitespace between the HTML comment delimiters and the tag, like this, '<!−− [tagname] −−>'.

7. The main tag parser is called.
   - Some tags parse recursively (depending on reparse and interpolate settings, of course).
   - Some tags (e.g., [loop]) process *prefix*−tags in their contained body text. Hence, the *prefix*−tags are not handled recursively.
   - Some tags are interpreted in the lib/Vend/Parse.pm:start routine. You cannot call them with the '$Tag−>*tagname*' syntax. They are:
     - ◊ The [goto] tag. Note also that the goto tag handles the [label] tag.
     - ◊ The [bounce] tag.

8. Image paths substitution on the HTML output occurs.

D.2. Nonstandard parsing within the admin interface

Parsing of content via the specialized regenerate usertag included with the administrative interface does not obey the above order. The MV_AUTOLOAD and '<!−−[tagname]−−>' escapes are skipped. There are some other more subtle differences as well; in the very unlikely event that you need to check this in the source

code, compare the 'interpolate_html' and 'cache_html' routines in Interpolate.pm.

D.3. Nonstandard parsing of Subtags

Subtags are parsed within the containing array−list or hash−list context created by the containing tag (see Looping tags and Sub−tags).

- All subtags at an earlier precedence level are treated before any in the next level.
- Within the same level, tags are proccessed in the order the appear on the page.
- Any standard tags are processed during 'interpolate' (before) or 'reparse' (after) phases of processing the containing tag.

Technical note

Processing within a hash− or array−list is actually done as a series of global regular expression substitutions on the page. Each substitution replaces one tag with the output of the subroutine(s) associated with it.

In array−list context, subtags are processed in the following order:

1. Check for *prefix*_**line** and prepare for it if present (does not process *prefix*−**line** at this time)
2. *prefix*−**sub** definitions processed
3. **if**−*prefix*−*etc.* nesting resolved
4. *prefix*−**alternate** processed
5. **if**−*prefix*−**param** processed
6. **if**−*prefix*−**pos** processed
7. *prefix*−**pos** processed
8. **if**−*prefix*−**field** processed
9. *prefix*−**line** processed
10. *prefix*−**increment** processed
11. *prefix*−**accessories** processed
12. *prefix*−**options** processed
13. *prefix*−**code** processed
14. *prefix*−**description** processed
15. *prefix*−**field** processed
16. *prefix*−**price** processed
17. *prefix*−**change** processed
18. *prefix*−**calc** processed
19. *prefix*−**exec** processed
20. *prefix*−**filter** processed
21. *prefix*−**last** processed
22. *prefix*−**next** processed
23. User's previous HTML widget SELECTED settings restored
24. Reparse standard tags in output of above (if reparse enabled for the containing tag)

In hash−list context, subtags are processed in the following order:

1. *prefix*−**sub** definitions processed
2. **if**−*prefix*−*etc.* nesting resolved
3. *prefix*−**alternate** processed
4. *prefix*−**line** processed

5. **if**−*prefix*−**param** processed
6. **if**−*prefix*−**field** processed
7. **if**−*prefix*−**modifier** processed (**if**−*prefix*−**param** and **if**−*prefix*−**modifier** are functionally identical except for parse order)
8. *prefix*−**increment** processed
9. *prefix*−**accessories** processed
10. *prefix*−**options** processed
11. *prefix*−**sku** processed
12. *prefix*−**code** processed
13. *prefix*−**quantity** processed
14. *prefix*−**modifier** processed
15. *prefix*−**param** processed
16. *prefix*−**quantity−name** processed
17. *prefix*−**modifier−name** processed
18. *prefix*−**subtotal** processed
19. *prefix*−**discount−subtotal** processed
20. *prefix*−**code** processed again differently (operating on new instances of *prefix*−**code** in output of above?)
21. *prefix*−**field** processed
22. *prefix*−**description** processed
23. *prefix*−**price** processed
24. *prefix*−**discount−price** processed
25. *prefix*−**difference** processed
26. *prefix*−**discount** processed
27. *prefix*−**change** processed
28. *prefix*−**tag** processed (*** CHECK THIS TAG NAME ***)
29. *prefix*−**calc** processed
30. *prefix*−**exec** processed
31. *prefix*−**filter** processed
32. *prefix*−**last** processed
33. *prefix*−**next** processed
34. User's previous HTML widget SELECTED settings restored
35. Reparse standard tags in output of above (if reparse enabled for the containing tag)

E. Search and Form Variables

E.1. Variable Names

E.1..1. other

Name	*scan*	*Type*	*Description*
mv_all_chars	ac	S	Turns on punctuation matching
mv_arg[0−9]+		A	Parameters for mv_subroutine (mv_arg0,mv_arg1,...)
mv_base_directory	bd	S	Sets base directory for search file names
mv_begin_string	bs	S	Pattern must match beginning of field
mv_case	cs	S	Turns on case sensitivity
mv_cartname		O	Sets the shopping cart name
mv_cache_params		S	Determines caching of searches
mv_change_frame		A	Any form, changes frame target of form output
mv_check		A	Any form, sets multiple user variables after update
mv_checkout		O	Sets the checkout page
mv_click		A	Any form, sets multiple form variables before update
mv_click		XA	Default mv_click routine, click is mv_click_arg
mv_click *name*		XA	Routine for a click *name*, sends click as arg
mv_click_arg		XA	Argument name in scratch space
mv_coordinate	co	S	Enables field/spec matching coordination
mv_column_op	op	S	Operation for coordinated search
mv_credit_card*		O	Discussed in order security (some are read−only)
mv_delay_page	dp	S	Delay search until after inital page display
mv_dict_end	de	S	Upper bound for binary search
mv_dict_fold	df	S	Non−case sensitive binary search
mv_dict_limit	di	S	Sets upper bound based on character position
mv_dict_look	dl	S	Search specification for binary search
mv_dict_order	do	S	Sets dictionary order mode
mv_doit		A	Sets default action
mv_email		O	Reply−to address for orders
mv_exact_match	em	S	Sets word−matching mode
mv_failpage	fp	O,S	Sets page to display on failed order check/search
mv_field_file	ff	S	Sets file to find field names for Glimpse
mv_field_names	fn	S	Sets field names for search, starting at 1
mv_first_match	fm	S	Start displaying search at specified match
mv_head_skip	hs	S	Sets skipping of header line(s) in index
mv_index_delim	id	S	Delimiter for search fields (TAB default)

mv_matchlimit	ml	S	Sets match page size
mv_max_matches	mm	S	Sets maximum match return (only for Glimpse)
mv_min_string	ms	S	Sets minimum search spec size
mv_negate	ne	S	Records NOT matching will be found
mv_nextpage	np	A	Sets next page user will go to
mv_numeric	nu	S	Comparison numeric in coordinated search
mv_order_group		O	Allows grouping of master item/sub item
mv_order_item		O	Causes the order of an item
mv_order_number		O	Order number of the last order (read−only)
mv_order_quantity		O	Sets the quantity of an ordered item
mv_order_profile		O	Selects the order check profile
mv_order_receipt		O	Sets the receipt displayed
mv_order_report		O	Sets the order report sent
mv_order_subject		O	Sets the subject line of order email
mv_orsearch	os	S	Selects AND/OR of search words
mv_profile	mp	S	Selects search profile
mv_range_alpha	rg	S	Sets alphanumeric range searching
mv_range_look	rl	S	Sets the field to do a range check on
mv_range_max	rx	S	Upper bound of range check
mv_range_min	rm	S	Lower bound of range check
mv_record_delim	dr	S	Search index record delimiter
mv_return_all	ra	S	Return all lines found (subject to range search)
mv_return_delim	rd	S	Return record delimiter
mv_return_fields	rf	S	Fields to return on a search
mv_return_file_name	rn	S	Set return of file name for searches
mv_return_spec	rs	S	Return the search string as the only result
mv_save_session		C	Set to non−zero to prevent expiration of user session
mv_search_field	sf	S	Sets the fields to be searched
mv_search_file	fi	S	Sets the file(s) to be searched
mv_search_line_return	lr	S	Each line is a return code (loop search)
mv_search_match_count		S	Returns the number of matches found (read−only)
mv_search_page	sp	S	Sets the page for search display
mv_searchspec	se	S	Search specification
mv_searchtype	st	S	Sets search type (text, glimpse, db or sql)
mv_separate_items		O	Sets separate order lines (one per item ordered)
mv_session_id	id	A	Suggests user session id (overridden by cookie)
mv_shipmode		O	Sets shipping mode for custom shipping
mv_sort_field	tf	S	Field(s) to sort on

mv_sort_option	to	S	Options for sort
mv_spelling_errors	er	S	Number of spelling errors for Glimpse
mv_substring_match	su	S	Turns off word−matching mode
mv_successpage		O	Page to display on successful order check
mv_todo		A	Common to all forms, sets form action
mv_todo.map		A	Contains form imagemap
mv_todo.checkout.x		O	Causes checkout action on click of image
mv_todo.return.x		O	Causes return action on click of image
mv_todo.submit.x		O	Causes submit action on click of image
mv_todo.x		A	Set by form imagemap
mv_todo.y		A	Set by form imagemap
mv_unique	un	S	Return unique search results only
mv_value	va	S	Sets value on one−click search (va=var=value)

E.2. Abbreviations

The two−letter abbreviations are mapped with these letters:

Abbr	***Long name***
DL	mv_raw_dict_look
MM	mv_more_matches
SE	mv_raw_searchspec
ac	mv_all_chars
ar	mv_arg
bd	mv_base_directory
bs	mv_begin_string
ck	mv_cache_key
co	mv_coordinate
cs	mv_case
cv	mv_verbatim_columns
de	mv_dict_end
df	mv_dict_fold
di	mv_dict_limit
dl	mv_dict_look
do	mv_dict_order
dp	mv_delay_page
dr	mv_record_delim
em	mv_exact_match
er	mv_spelling_errors
fi	mv_search_file

fm	mv_first_match
fn	mv_field_names
hs	mv_head_skip
id	mv_session_id
il	mv_index_delim
ix	mv_index_delim
lb	mv_search_label
lo	mv_list_only
lr	mv_line_return
lr	mv_search_line_return
ml	mv_matchlimit
mm	mv_max_matches
mp	mv_profile
ms	mv_min_string
ne	mv_negate
np	mv_nextpage
nu	mv_numeric
op	mv_column_op
os	mv_orsearch
pc	mv_pc
ra	mv_return_all
rd	mv_return_delim
rf	mv_return_fields
rg	mv_range_alpha
rl	mv_range_look
rm	mv_range_min
rn	mv_return_file_name
rr	mv_return_reference
rs	mv_return_spec
rx	mv_range_max
se	mv_searchspec
sf	mv_search_field
si	mv_search_immediate
sp	mv_search_page
sq	mv_sql_query
st	mv_searchtype
su	mv_substring_match
tf	mv_sort_field

to	mv_sort_option
un	mv_unique
va	mv_value

line:

Interchange Upgrade Guide

59. Introduction

This document contains, in rough form, notes on upgrading from Minivend 3 to Minivend 4, and Minivend 4 to Interchange.

60. Interchange 4.8 Deprecated Features

This document describes features of Interchange 4.8 that have been deprecated. Any use of these features should be discontinued. In most cases we have provided an alternative mechanism to accomplish the same results. These deprecated features may be removed at some point in the future. You should change to the new mechanism to avoid breakage.

60.1. Deprecated Features Previous to Interchange 4

This section needs some serious work.

cart/page from path

interchange.PL 308,313

```
            if($path =~ s:/(.*)::) {
                    $cart = $1;
                    if($cart =~ s:/(.*)::) {
                            $page = $1;
                    }
            }
```

mv_orderpage

interchange.PL 321,323

```
            $CGI::values{mv_nextpage} = $CGI::values{mv_orderpage}
                                                                    || find_special_page('order')
                    if ! $CGI::values{mv_nextpage};
```

$decode

interchange.PL 493

```
                            HTML::Entities::decode($value) if $decode;
```

mv_orderpage

interchange.PL 854,855

```
                                            $CGI::values{mv_nextpage} = $CGI::values{mv_orderpage
                                                    if $CGI::values{mv_orderpage};
```

ROUTINES and LANG

intechange.PL 1552,1579

```
            ROUTINES: {
                    last ROUTINES unless index($Vend::FinalPath, '/process/') == 0;
                    while ($Vend::FinalPath =~ s:/process/(locale|language|currency)/([^/]*)/:/pr
                            $::Scratch->{"mv_$1"} = $2;
                    }
                    $Vend::FinalPath =~ s:/process/page/:/:;
```

```
}
my $locale;
if($locale = $::Scratch->{mv_language}) {
        $Global::Variable->{LANG}
                = $::Variable->{LANG} = $locale;
}

if ($Vend::Cfg->{Locale}
        $locale = $::Scratch->{mv_locale}       and
        defined $Vend::Cfg->{Locale_repository}->{$locale}
        )
{
        $Global::Variable->{LANG}
                        = $::Variable->{LANG}
                        = $::Scratch->{mv_language}
                        = $locale
                 if ! $::Scratch->{mv_language};
        Vend::Util::setlocale(  $locale,
                                                                ($::Scratch->{mv_currency} ||
                                                                { persist => 1 }
}
```

list_compat

lib/Vend/Interpolate.pm 2808

```
list_compat($opt->{prefix}, \$text);
```

lib/Vend/Interpolate.pm 3538

```
list_compat($opt->{prefix}, \$text);
```

lib/Vend/Interpolate.pm 3874

```
list_compat($opt->{prefix}, \$page);
```

find_sort

lib/Vend/Interpolate.pm 3270,3271

```
$text =~ /^\s*\[sort\s+.*/si
        and $opt->{sort} = find_sort(\$text);
```

mv_order_report

lib/Vend/Order.pm 867,868

```
$body = readin($::Values->{mv_order_report})
            if $::Values->{mv_order_report};
```

mv_error_$var

lib/Vend/Order.pm 1030

```
                        $::Values->{"mv_error_$var"} = $message;
```

60.2. Interchange 4 Deprecated Features

Vend::Util::send_mail Vend::Order::send_mail send_mail

The send_mail routine has been replaced by the Vend::Mail::send routine.

61. Upgrading from Minivend 4.0 to Interchange 4.6

if [item−price] suddenly turns 0, check PriceField in the catalog.cfg

61.1. minivend.cfg

- Remove references to MiniMate.
- Add this line to minivend.cfg:
  #include lib/UI/ui.cfg
  Make sure the files catalog_before.cfg and catalog_after.cfg are there, or add their contents to etc/your_cat_name.before and etc/your_cat_name.after to it only for some catalogs.

61.2. Access Manager

You need to get the minimate.asc file renamed to access.asc and add the following fields to the first line:

```
groups
last_login
name
password
```

Remove these catalog.cfg lines:

```
Variable            MINIMATE_META   mv_metadata
Variable            MINIMATE_TABLE  minimate
Database            minimate        minimate.asc      TAB
```

Add this one:

```
Database            affiliate    affiliate.txt     TAB
```

Authentication for admin users is now done from a separate table than customers, and passwords are encrypted.

61.3. Database Editing

Update the mv_metadata.asc file as appropriate.

61.4. Order Manager

Some things that are needed for the order manager:

- Add these fields to transactions:

```
affiliate  approx. char(32)
archived   char(1)
campaign   approx. char(32)
comments   blob/text
complete   char(1)
deleted    char(1)
order_wday char(10)
order_ymd  char(8)
```

```
po_number  approx. char(32)
```

- Add these fields to transactions:

```
affiliate  approx. char(32)
campaign   approx. char(32)
```

- Remove this field from userdb:

```
mv_credit_card_info
```

- Add these fields to userdb:

```
inactive     char(1)
credit_limit char(14) or decimal(12,2)
dealer       char(3)
```

- Create the directory 'logs'.
- Create the directory 'orders' if it doesn't already exist.
- Update your order routes to those in the Interchange distribution. Note that the route log_entry is necessary if you want to enter orders from the Interchange UI.
- Update the etc/log_transaction file.
- Add the etc/log_entry file.
- Add this to catalog.cfg:

```
## Don't want people setting their credit_limit directly
UserDB default scratch "credit_limit dealer"
```

61.5. Affiliates

Add a tab−delimited affiliate table:

```
affiliate name    campaigns   join_date   url timeout active  password
```

You can find a recommended database configuration in foundation/dbconf/*/affiliate.*.

61.6. Page Editor

Add the directories 'templates' and 'backup'. Copy the contents of the Interchange simple/templates to templates.

61.7. Item Editor

Add a merchandising table with the following fields:

```
Database  merchandising  merchandising.txt __SQLDSN__
Database  merchandising  DEFAULT_TYPE text

sku                 char(32)
featured            char(32)
banner_text
banner_image
```

```
blurb_begin
blurb_end
timed_promotion     char(16)
start_date          char(24)
finish_date         char(24)
upsell_to
cross_sell
cross_category      char(64)
others_bought
times_ordered
```

Index the fields with char(*) types. You can find the recommended database configuration in foundation/dbconf/*/merchandising.*

61.8. Preferences Editor (KNAR)

Create the tab−delimited file variable.txt with these fields:

```
code  Variable  pref_group
```

Add this as the *first* line of catalog.cfg:

```
VariableDatabase variable
```

61.9. Route Editor

Create the file route.txt with these fields:

```
code
report
receipt
encrypt_program
encrypt
pgp_key
pgp_cc_key
cyber_mode
credit_card
profile
inline_profile
email
attach
counter
increment
continue
partial
supplant
track
errors_to
```

Add this line in catalog.cfg:

```
RouteDatabase route
```

61.10. Transactions database

The back office UI should work fine for editing database tables. Obviously the things which are specific to the order transaction setup will break unless you have the right fields, but even these can be controlled by configuring the UI.
Add a new field to transaction.txt called 'archived'.

62. Upgrading from Minivend 3 to Minivend 4

There were big changes from Minivend 3 to Minivend 4, some of which were incompatible.
Many things were removed as redundant, deprecated, or just plain crufty:

62.1. Nested [loop]s

MV3 used a different scheme for creating nested loop lists:
[loop with="−a"* arg="item item item" search="se=whatever"]
allowed you to refer to the nested values with a [loop−code−a] construct. In Minivend 4, the form is:

```
[loop prefix=size list="Small Medium Large"]
        [loop prefix=color list="Red White Blue"]
                [color-code]-[size-code]<BR>
        [/loop]
        <P>
[/loop]
```

62.2. All frame features removed

Frames are now managed by the user in HTML.

62.3. Tags removed

62.3.1. buttonbar

Replace with Variable defined in catalog.cfg. buttonbar was previously used as an SSI−like command for catalog−wide standardized features like navigation bars. In the 3.x catalog.cfg the ButtonBars parameter defines a list of html snippets, like

```
ButtonBars header.html footer.html copyright.html
```

So [buttonbar 0] substitutes 'header.html', [buttonbar 1] substitutes 'footer.html', etc.
In 4.x catalog.cfg, define variables, like

```
Variable HEADER    <pages/header
Variable FOOTER    <pages/footer
Variable COPYRIGHT <pages/copyright
```

Then replace all occurences of [buttonbar 0] with __HEADER__, [buttonbar 1] with __FOOTER__, etc.
Note that the old header.html, footer.html, etc. contained html code, but were not actually html pages with <html><body> etc, tags. Thus the current practice is to use filenames with no extension or perhaps '.txt' to differentiate them from pages.

62.3.2. random

Replace with [ad random=1] or custom code. See the [ad] tag docs. Random and rotate were used to place random or rotating regions on pages, such as banner ads.
The Random durective in catalog.cfg defines the numbered HTML snippet files, similar to buttonbars above.

62.3.3. rotate

Replace with [ad ...]. See [random] above.

62.3.4. help

No replacement. Use data functions or variables.

62.3.5. body

Replace with templates. Again the body tag [body 1] etc. defines numbered body definitions that could be applied site−wide. However, in this case minivend actually built up the <body ....> substitution using the Mv_* directives in catalog.cfg.

62.3.6. finish_order

[finish_order] was a conditional tag; if the basket contained anything a 'checkout' graphic would be displayed. No replacement; use [if items]Message[/if].

62.3.7. last_page

No replacement − this can be emulated by setting a scratch variable on one page, then using it to build the return URL.

62.3.8. item−link

No replacement, just use [page [item−code]].

62.3.9. loop−link

No replacement, just use [page [loop−code]].

62.3.10. sql−link

No replacement, just use [page [sql−code]].

62.3.11. accessories

Replace with normal data functions.

62.3.12. Compatibility routines

Compatibility routines for many popular tags like [random], [rotate], etc. are provided in the appendix of this document. To use one, copy it to a file and put it in your usertag directory. (Tags in the usertag directory are read in by interchange.cfg by default).

62.4. Directives removed

```
ActionMap
AdminDatabase
AdminPage
AsciiBackend
BackendOrder
ButtonBars
CheckoutFrame
CheckoutPage
CollectData
DataDir
Delimiter
DescriptionTrim
FieldDelimiter
FrameFlyPage
FrameLinkDir
FrameOrderPage
FrameSearchPage
ItemLinkDir
ItemLinkValue
MsqlDB
MsqlProducts
Mv_AlinkColor
Mv_Background
Mv_BgColor
Mv_LinkColor
Mv_TextColor
Mv_VlinkColor
NewReport
NewTags
OldShipping
OrderFrame
PageCache
PriceDatabase
Random
ReceiptPage
RecordDelimiter
ReportIgnore
Rotate
SearchFrame
SearchOverMsg
SecureOrderMsg
SpecialFile
SubArgs
Tracking
```

62.5. Minor operations removed

- auto−substitution of mp= on [loop search=profile], [search−region arg=profile]
- [tag scan]...
- [tag sql]...

  Many of these are related to one of:
- Removal of frames logic
- Removed tags
- Obsolete methods
- Old routines for 2.0x compatibility

62.6. Search lists

Search tags must now be surrounded by [search−region] [/search−region]. This is because multiple searches can be done in a page, with multiple [more−list] entries, multiple [no−match] areas, etc. It was not really possible to avoid this and add the feature.
To find all files containing the search list, do:

```
find pages −type f | xargs grep −l '\[search.list'
```

That will yield a set of files that need to be updated. You should surround all parts of the search area, i.e.:

```
[search−region]

[search−list]
    your search iteration stuff, [item−code], etc.
[/search−list]

[more−list]
    [more]
[/more−list]

[/search−region]
```

62.7. Search conditionals

Search conditionals should now say [if−item−field field] [/if−item−field] and [if−item−data table column] [/if−item−data]. This allows mixing and nesting of lists. You may find that the old works in some situations, but it will not work in all situations.

62.8. Form data updates

Added Scratch variable mv_data_enable to gate the update_data function. You must set it before doing a form update. Prior to this it was possible to update a SQL database willy−nilly.
A quick fix like this will allow the update on a single page:

```
[set update_database]
[set mv_data_enable]1[/set]
[/set]
<INPUT TYPE=hidden NAME=mv_click VALUE=update_database>
```

It will ensure at least that the user loads one form from you for each update. For best security, gate with a userdb entry like this:

```
[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]
```

62.9. Checkout changes

Minivend 4 uses in−page error−flagging on the checkout page. Simplest way to convert is probably to use the checkout.html from the simple demo as a start, and move in any customization from the existing site's catalog.html (headers, footers, logos, etc.) A line−by−line comparison of the data fields in the checkout page should be performed, adding any custom fields as needed. Custom error checking in etc/order.profiles may have to be re−worked, or can be added into checkout.html using the in−page order profile capability. Remember to update receipt.html and report/report.html with any custom fields, as well.

62.10. [if−field] etc.

The least−compatible things in the tag area are [if−field] (needs to be [if−PREFIX−field], where prefix might be item|loop by default depending on the tag. Likewise:

```
[if-data table col]  --> [if-PREFIX-data table col]
[on-change mark]     --> [PREFIX-change mark]
[if-param param]     --> [if-PREFIX-param param]
[PREFIX-param N]     --> [PREFIX-pos N] (where N is a digit)
```

62.11. [search−list]

You must always surround [search−list] with [search−region] [/search−region].
Embedded Perl changes quite a bit. While there are the $Safe{values} and other variable settings, they are automatically shared (no arg="values") and move to:

```
$Safe{values}   -->  $Values
$Safe{cgi}      -->  $CGI
$Safe{carts}    -->  $Carts
$Safe{items}    -->  $Items
$Safe{config}   -->  $Config
$Safe{scratch}  -->  $Scratch
```

There are a number of other objects, see the docs.
Most other issues have more to do with the catalog skeleton (i.e. simple or barry or basic or art) than they do the core. For instance, the "basic" catalog produced for MV3 ran unchanged except for the issues discussed above.

62.12. Global subs

Accessing globalsubs from [perl] tags is done slightly differently.
Minivend 3 method:

```
[perl sub]
myfunsub();
[/perl]
```

Minivend 4/IC method:

```
[perl subs=1]
myfunsub();
[/perl]
```

If you do this wrong, you'll get an error that looks like this:

```
115.202.115.237 H8gbq6oK:115.202.115.237 - [28/February/2001:18:58:50 -0500] testcat /cgi-bin
```

F. Minivend 3 compatibility usertags and globalsubs

These files were originally distributed with Minivend 4 in the compat/ directory. They replace Minivend 3 functionality that was removed or greatly altered in Minivend 4.

F.1. body

```
UserTag body PosNumber 2
UserTag body Order type extra
UserTag body Routine <<EOR
use vars qw($C);
sub parse_color {
    my ($var, $value) = @_;
    return '' unless $value;
    $var = lc $var;
    $C->{Color}->{$var} = [];
    @{$C->{'Color'}->{$var}} = split /\s+/, $value;
    return $value;
}

sub {
    my($scheme, $extra) = @_;
    my $r = '<BODY';
    my ($var,$tag);
    #return '<BODY>' unless (int($scheme) < 16 and int($scheme) > 1);

    my %color = qw( mv_bgcolor BGCOLOR mv_textcolor TEXT
                    mv_linkcolor LINK mv_vlinkcolor VLINK
                    mv_alinkcolor ALINK mv_background BACKGROUND );
    if (defined $::Values->{mv_resetcolors}
            and $::Values->{mv_resetcolors}) {
        delete $::Values->{mv_customcolors};
        undef $::Values->{mv_resetcolors};
    }
    if (defined $::Values->{mv_customcolors}) {
        foreach $var (keys %color) {
            $r .= qq| $color{$var}="| . $::Values->{$var} . '"'
                if $::Values->{$var};
        }
    }
    else {
        foreach $var (keys %color) {
            $r .= qq| $color{$var}="| . ${$Vend::Cfg->{Color}->{$var}}[$scheme] . '"'
                if defined ${$Vend::Cfg->{Color}->{$var}}[$scheme]
                    &&  ${$Vend::Cfg->{Color}->{$var}}[$scheme] !~ /\bnone\b/;
        }
    }
    $r =~ s#(BACKGROUND="(?!http:))([^/])#$1$Vend::Cfg->{ImageDir}$2#;
    $r .= " $extra" if defined $extra;
    $r .= '>';
}
EOR

AddDirective Mv_Background   color
AddDirective Mv_BgColor      color
AddDirective Mv_TextColor    color
AddDirective Mv_LinkColor    color
```

```
AddDirective Mv_AlinkColor   color
AddDirective Mv_VlinkColor   color
```

F.2. buttonbar

```
# Returns a buttonbar by number
UserTag buttonbar Order type
UserTag buttonbar PosNumber 1
UserTag buttonbar Interpolate 1
UserTag buttonbar Routine <<EOR
sub get_files {
    my($dir, @files) = @_;
    my(@out);
    my($file, $contents);
    foreach $file (@files) {
        config_error(
          "No leading ../.. allowed if NoAbsolute set. Contact administrator.\n")
        if $file =~ m#^\.\./.*\.\.# and $Global::NoAbsolute;
        push(@out,"\n") unless
            push(@out,readfile("$dir/$file.html"));
    }

    @out;
}

sub parse_buttonbar {
    my ($var, $value) = @_;
    return [] unless $value;
    my @c;
    my @vals = grep /\S/, split /\s+/, $value;
    for(@vals) {
        push @c, Vend::Util::readfile("pages/$_.html");
    }
    return \@c;
}

sub {
    my($buttonbar) = @_;
    if (defined $Vend::Cfg->{'ButtonBars'}->[$buttonbar]) {
        return $Vend::Cfg->{'ButtonBars'}->[$buttonbar];
    }
    else {
        return '';
    }
}
EOR

AddDirective ButtonBars  buttonbar
```

F.3. form_mail.cfg

```
GlobalSub <<EndOfSub
sub form_mail {
    my($to, $subject, $reply, $body) = @_;
    my($ok);

    $subject = '<no subject>' unless defined $subject && $subject;

    $reply = '' unless defined $reply;
```

```
    $reply = "Reply-to: $reply\n" if $reply;

    $ok = 0;
    SEND: {
        open(Vend::MAIL,"|$Vend::Cfg->{'SendMailProgram'} -t") or last SEND;
        print Vend::MAIL
            "To: $to\n",
            $reply,
            "Subject: $subject\n",
            "Errors-To: $Vend::Cfg->{MailOrderTo}\n\n",
            $body
            or last SEND;
        close Vend::MAIL or last SEND;
        $ok = ($? == 0);
    }

    if (!$ok) {
        logError("Unable to send mail using $Vend::Cfg->{'SendMailProgram'}\n" .
            "To '$to'\n" .
            "With subject '$subject'\n" .
            "With reply-to '$reply'\n" .
            "And body:\n$body");
    }
    $ok;
}
EndOfSub
```

F.4. help

```
UserTag help PosNumber 1
UserTag help Order name
UserTag help Routine <<EOR
sub parse_help {
    my ($var, $value) = @_;
    my (@files);
    my (@items);
    my ($c, $chunk, $item, $help, $key);
    unless (defined $value && $value) {
        $c = {};
        return $c;
    }
    $c = $C->{'Help'};
    $var = lc $var;
    $C->{'Source'}->{'Help'} = $value;
    @files = get_files($C->{'PageDir'}, split /\s+/, $value);
    foreach $chunk (@files) {
        @items = split /\r?\n\r?\n/, $chunk;
        foreach $item (@items) {
            ($key,$help) = split /\s*\n/, $item, 2;
            if(defined $c->{$key}) {
                $c->{$key} .= $help;
            }
            else {
                $c->{$key} = $help;
            }

        }
    }
    return $c;
}
```

```
sub {
    my($help) = shift;
    # Move this to control section?
    if ($::Values->{mv_helpon}) {
        delete $::Values->{mv_helpoff};
        undef $::Values->{mv_helpon};
    }
    return '' if defined $::Values->{'mv_helpoff'};
    if (defined $Vend::Cfg->{'Help'}{$help}) {
        return $Vend::Cfg->{'Help'}{$help};
    }
    else {
        return '';
    }
}
EOR

AddDirective Help help
```

F.5. random_rotate

```
UserTag random PosNumber 0
UserTag random Interpolate 1
UserTag random Routine <<EOR
package Vend::Config;
sub parse_random {
    my ($var, $value) = @_;
    return '' unless (defined $value && $value);
    my $c = [];
    $var = lc $var;
    my @files = grep /\S/, split /\s+/, $value;
    local ($Vend::Cfg) = $C;
    for (@files) { push @$c, Vend::Util::readin($_) }
    return $c;
}

package Vend::Interpolate;
sub {
    my $random = int rand(scalar(@{$Vend::Cfg->{'Random'}}));
    if (defined $Vend::Cfg->{'Random'}->[$random]) {
        return $Vend::Cfg->{'Random'}->[$random];
    }
    else {
        return '';
    }
}
EOR

UserTag rotate PosNumber 2
UserTag rotate Order ceiling floor
UserTag rotate Interpolate 1
UserTag rotate Routine <<EOR
sub {
    return '' unless $Vend::Cfg->{Rotate};
    my $ceiling = $_[0] || @{$Vend::Cfg->{'Rotate'}} || return '';
    my $floor   = $_[1] || 1;

    $ceiling--;
    $floor--;

    my $marker = "rotate$floor$ceiling";
```

```
    if($ceiling < 0 or $floor < 0) {
        $floor = 0;
        $ceiling = scalar  @{$Vend::Cfg->{'Rotate'}} - 1;
        logError "Bad ceiling or floor for rotate";
    }

    my $rotate;
    $rotate = $Vend::Session->{$marker} || $floor;

    if($rotate > $ceiling or $rotate < $floor ) {
        $rotate = $floor;
    }

    $Vend::Session->{$marker} = $rotate + 1;
    return $Vend::Cfg->{'Rotate'}->[$rotate];
}
EOR

AddDirective Random random
AddDirective Rotate random
```

F.6. AsciiBackend

```
GlobalSub <<EOS
sub AsciiBackend {
    package Vend::Order;
    $Vend::Order::override_track_order = \&track_order;
    sub track_order_backend {
        my ($order_no,$order_report) = @_;
        my ($c,$i);
        my (@backend);

        @backend = split /\s*,\s*/, $Vend::Cfg->{BackendOrder};

        if(@backend and $Vend::Cfg->{AsciiBackend}) {
            my(@ary);
            push @ary, $order_no;
            for(@backend) {
                push @ary, $::Values->{$_};
            }
            foreach $i (0 .. $#$Vend::Items) {
                push @ary, $Vend::Items->[$i]{'code'};
                push @ary, $Vend::Items->[$i]{'quantity'};
                if ($Vend::Cfg->{UseModifier}) {
                    foreach $j (@{$Vend::Cfg->{UseModifier}}) {
                        push @ary, $Vend::Items->[$i]->{$j}
                    }
                }
            }
            logData ($Vend::Cfg->{AsciiBackend}, @ary);
        }
        $Vend::Order::override_track_order->($order_no, $order_report);
    }
    *track_order = \&Vend::Order::override_track_order;
}
EOS

AddDirective BackendOrder
AddDirective AsciiBackend
```

63. History of Interchange

Interchange is a descendent of Vend, an e−commerce solution originally developed by Andrew Wilcox in early 1995. Mike Heins took the first publicly−released version, Vend 0.2, and added searching and DBM catalog storage to create MiniVend. Mike released MiniVend 0.2m7 on December 28, 1995. Subsequent versions of MiniVend took parts from Vend 0.3, especially the vlink and Server.pm modules, which were adapted to run with MiniVend. In the four years that followed, Mike Heins expanded and enhanced MiniVend, creating a powerful and versatile e−commerce development platform. MiniVend grew to support thousands of businesses and their e−commerce sites.

Separately, an experienced e−commerce development team founded Akopia. Their goal was to create a sophisticated open source e−commerce platform that was both feature−rich and easy to use. Their product, Tallyman, was intuitive, and had great content−management features, but lacked many of MiniVend's capabilities.

Akopia acquired MiniVend in June 2000. Mike Heins and the Tallyman developers combined MiniVend with Tallyman's features to create Interchange. Interchange replaces both MiniVend and Tallyman. In order to preserve compatibility, the name "minivend" and prefixes like "mv_" and "MVC_" will still appear in source code and configuration files.

In January 2001, Red Hat acquired Akopia and created its new E−Business Solutions Division. Interchange development is going forward and the user community continues to grow. line:

Interchange + CVS HOWTO

64. Introduction

64.1. Preamble

64.2. Purpose

The purpose of this document is to help others take advantage of CVS and Interchange together to increase the quality of their programming, whether they are sole developers or part of a large team of programmers, graphic artists, and HTML design gurus. Portions of it apply to general CVS setup and use, but it is geared toward the average developer using Interchange to implement an e−commerce website.

64.3. Audience

I intend for this document to be useful to those who are not yet familiar with CVS as well as those who are. If you already know how to setup a pserver then you might just skim chapter 2 ("Setup CVS"), or just skip it all together.
In addition, I have tried to write at a technical level that would be on par with what I percieve to be the average Interchange user that participates in the interchange−users mailing list. It is assumed that the reader can and has setup Interchange and the foundation (or construct) template catalog is working correctly.

64.4. Contact the author

If you find any spelling errors, technical slipups, mistakes, subliminal messages, or if you wish to send feedback, critique, remarks, comments, or if you wish to contribute examples, instructions for alternative platforms, chapters, or other material, please do so.
The preferred method of submitting these changes is in the form of a context diff against the SDF source file (ic_cvs.sdf). Please address your correspondence to:
Dan Browning danpb@mail.com

64.5. The advantages of using CVS

CVS is a very useful tool and can help you in your development, no matter if you are one developer or are coordinating a team of developers.

- What is CVS all about?
- What are it's advantages?
  The official CVS website (http://www.cvshome.org/new_users.html) has more detailed answers to these questions, but here are some brief points of interest.
- Checkout "historic" points in time or milestones in a project, for example when an e−commerce site went "live" or before a major branch in the code.
- Revert to older versions of a file, directory, or an entire website.
- Branching releases. Concurrently develop an unstable development version as well as fix bugs in the stable production version.
- Multiple developers can work on the same catalog and even the same file at the same time. (For more information about how multiple simultaneous writes are merged and conflicts resolved, see the cvs

docs in the Resources Appendix).

- CVS is better than ftp for file transfer, because it automatically downloads only changed files, and even then, only the portion of the file that has changed (using patches).
- CVS can automatically merge two simultaneous writes to the same file by different developers.
- Allows one to keep track of the changes that have been made over time (many release managers repackage cvs commit logs into WHATSNEW, HISTORY, and/or NEWS files).

64.6. How to use this document

There are many potential uses of CVS as it applies to Interchange. In fact, there are as many unique ways to use CVS as there are unique developers. This document can only cover some of the ways, including basic and useful techniques to get started using CVS. For the intents of the average web developer using IC for a B2C e−commerce site, I've identified a few of the possible uses:
Simple

- One server
- One catalog
- One cvs module
- One branch

Medium

- One server
- Two catalogs (one is live, one is development)
- One cvs modules
- Seperate development and live branches

Complex/Custom

- Multiple servers (e.g., developers' servers, staging servers, and live servers)
- Multiple catalogs
- Multiple cvs modules
- Multiple branches
- Custom setup

This document attempts to cover the Simple well, explain many aspects of the Medium, which will hopefully give you the background you need if you decide to setup your own complex development environment.

65. Setup CVS

65.1. Assumptions

Here are some of the assumptions that I make that apply to various parts of the rest of this document:

- Red Hat 7.x
- Interchange installed (RPM, tarball, or CVS)
- Default interchange tarball directory paths (adjust for your environment)
- foundation catalog setup and working

Note: I will assume "foundation" for the catalog name and directory paths, but it should be just as easy to use this document with the construct catalog or your own catalog by mentally transposing the names and paths.

There shouldn't be any reason why you could not do everything I mention here on other Linux distributions, unicies or cygwin. However, my statements will reflect Red Hat 7.x. Additionally, Red Hat 6.x is for the most part the same as 7.x, except for the difference of using inetd instead of xinetd to setup pserver.

65.2. Installing CVS

This is the easy part. For Red Hat systems, download the cvs rpm's and install them. The following RPM command will download and install the Red Hat 7.1 version of cvs from rpmfind.net.

Note: You need to be root to complete the following tasks

```
su − root
rpm −Uvh ftp://speakeasy.rpmfind.net/linux/redhat/7.1/en/os/i386/RedHat/RPMS/cvs−1.11−3.i386.
```

Create the user and group that will administrate the interchange repository. For this document, it will be the interch user, (which has already been created by the RPM installation, if you use that). But if you understand the mechanics of Unix users/groups, then you can use whatever username and group scheme you prefer. For example, some create a cvs user and cvs group, then add the interchange user and catalog owner to it's group and/or vise−versa. The integration of interchange and CVS in the latter portion of this document will require that the CVS user has some write capability to the catalog directory.

65.3. Create the CVS repository directory

You will need to create a repository directory such as `/rep`, which is used here and in the rest of the document, but it can be any directory you desire, and must be owned by the cvs user. Many use `/var/rep` or `/home/cvs/rep`.

```
su − root
mkdir /rep
chown interch.interch /rep
```

65.4. Setup environment variables

The CVSROOT and EDITOR environment variables should be setup for all users in /etc/profile. Of course, EDITOR can be whatever Unix text editor you prefer, such as `vi`, `emacs`, `pico`, or `joe`.

`/etc/profile:`

```
export CVSROOT=/rep
export EDITOR=vi
```

Note: You will need to logout/login for the profile changes to take effect.

65.5. Initialize the repository

Initialize the repository as the CVS user, which is `interch` for this document.

```
su - interch
cvs -d /rep init
```

65.6. CVS Authentication

65.6.1. Background

Authentication is done in CVS through the `$CVSROOT/CVSROOT/passwd` file. It can be easily manipulated through some of the CVS administration tools that are available.

65.6.2. CVS administration tools

- http://freshmeat.net/projects/cvsadmin/
- http://freshmeat.net/projects/cvspadm/
  I recommend cvsadmin, but there is also a manual method that can be used in the absence of such tools, which involves copying the system shadow file and modifying it for use by CVS. For more information on the manual method, see the RedHat CVS pserver setup guide by Michael Amorose (http://www.michael-amorose.com/cvs/).

65.6.3. Setup authentication using the cvsadmin tool

You can find a tarball to install on your system using the above address, but here is the address of a recent RPM package of the version. This package is intended for mandrake systems, but is compatible with Red Hat 7.1:

- ftp://speakeasy.rpmfind.net/linux/Mandrake-devel/contrib/RPMS/cvsadmin-1.0.1-1mdk.i586.rpm
  After installing, create a password file (*touch $CVSROOT/CVSROOT/passwd*), and execute `cvsadmin add <usernames>` for each of your usernames.

65.7. Setup CVS modules

Note: From this point on, assume that all commands are executed as the CVS user (e.g. interch), unless otherwise specified.

A module is CVS is like the concept of a "project", where each module has its own branches, trees, and other features.

65.7.1. Add your project to `modules`

The format of the modules file is explained in detail in the cvs documentation, here is the simplest way to use it:

`/rep/CVSROOT/modules:`

```
<Module name><TAB><Module Directory>
```

The module name can be whatever you want, and the module directory is what we will create later under /rep. We'll want a module for the template catalog (foundation). For example:

```
foundation      foundation
```

65.7.2. Create the module directory

This is the directory that is referred to in the `CVSROOT/modules` file we just modified.

```
mkdir /rep/foundation
```

65.8. Setup binary file types

This isn't necessary if you aren't going to manage any binary files (e.g. if you plan on excluding your /images/ directory). But I recommend including it. The following is an example including many binary file types (by extension) used in web development.

`/rep/CVSROOT/cvswrappers:`

```
*.avi   -k 'b' -m 'COPY'
*.doc   -k 'b' -m 'COPY'
*.exe   -k 'b' -m 'COPY'
*.gif   -k 'b' -m 'COPY'
*.gz    -k 'b' -m 'COPY'
*.hqx   -k 'b' -m 'COPY'
*.jar   -k 'b' -m 'COPY'
*.jpeg  -k 'b' -m 'COPY'
*.jpg   -k 'b' -m 'COPY'
*.mov   -k 'b' -m 'COPY'
*.mpg   -k 'b' -m 'COPY'
*.pdf   -k 'b' -m 'COPY'
*.png   -k 'b' -m 'COPY'
*.ppt   -k 'b' -m 'COPY'
*.sit   -k 'b' -m 'COPY'
*.swf   -k 'b' -m 'COPY'
*.tar   -k 'b' -m 'COPY'
*.tgz   -k 'b' -m 'COPY'
*.tif   -k 'b' -m 'COPY'
*.tiff  -k 'b' -m 'COPY'
*.xbm   -k 'b' -m 'COPY'
*.xls   -k 'b' -m 'COPY'
*.zip   -k 'b' -m 'COPY'
```

65.9. Testing your repository

At this point, you should have a working (though empty) CVS repository. Before we continue with setting up the pserver or importing source code, try logging in as one of the cvs users listed in your CVSROOT/passwd and test the checkout.

```
#test checkout in home directory of any cvs user
mkdir ~/src
cd ~/src
cvs co foundation
```

This should create `foundation/` and `foundation/CVS`.

65.10. Setup the CVS pserver

You will likely need to be root to do this, and there are lots of guides on the internet for setting up a cvs pserver, hopefully you wont have any trouble doing it on your particular operating system. See the Resources Appendix for more information.

65.10.1. Setup pserver in Red Hat 7.1 using xinetd.

For Red Hat 7.x, edit `/etc/xinetd.d/cvspserver` (create a new one if none exists). The following works for me, but customization may be required for your environment (see the next section below for an inetd−based system example). This also must be done as root.

```
su − root
/etc/xinetd.d/cvspserver:

# default: on
service cvspserver
{
        disable = no
        socket_type  = stream
        protocol  = tcp
        wait   = no
        user    = root
        server   = /usr/bin/cvs
        server_args  = −f −−allow−root=/rep pserver

}
```

Also unset the HOME variable in xinetd. This was required for my repository to work correctly, but if anyone has a better suggestion, I would appreciate a note.

```
/etc/xinetd.d/cvspserver:

unset HOME
```

Now, restart xinetd for the changes to take effect.

```
service xinetd restart
```

65.10.2. Setup pserver in inetd−based systems.

I haven't tested this (any takers?), but something like the following needs to be done for inetd−based systems such as Red Hat 6.2. Make sure that the following files are setup accordingly.

/etc/services:

```
cvspserver      2401/tcp
```

N:/etc/inetd.conf:

```
cvspserver stream tcp nowait root /usr/sbin/tcpd /usr/local/bin/cvs −−allow−root=/usr/local/n
```

65.10.3. Testing your pserver

At this point, you should be able to use a cvs client to use your pserver and execute all the same commands that you can locally (which we tested before). You may wish to take advantage of a graphical cvs client, which can be particularly helpful in leveling the learning curve.
See the Resources Appendix for links to some graphical CVS tools.

66. Import your Interchange catalog into CVS

66.1. Configuring your catalog

Eventually, we will import your catalog into the cvs repository, but first we need to do some work with a temporary copy of the catalog so we can get it into shape for importing.

Note: From here on, assume the use of the interchange user, such as `interch`, unless otherwise noted.

```
su – interch
```

If you installed via RPM:

```
service interchange stop
```

If you installed via tarball (default path):

```
/usr/local/interchange/bin/interchange ––stop
```

66.2. Remove old CVS folders

If, for any reason, you already have CVS files (all the `CVS/` directories)in your catalog, they must be removed because they might interfere with the new CVS setup. For example, maybe you moved servers and you are setting up CVS again. You might use the following `find` command, which will find any folders named `CVS` in the current directory and remove them. There is probably a better way to deal with old `CVS/` folders, but the following works for me (again, suggestions welcome).

Note: You should make a backup of the catalog directory before you do this.

```
#Become interchange catalog user
su – interch

#backup catalog folder first
tar czf ~/foundation_backup.tgz /var/.../foundation

#get rid of any old CVS folders –– (BE CAREFULL!)
cd /var/lib/interchange/foundation
find . –name CVS –exec rm –Rf {} \;
```

66.3. Create a working copy of your catalog

A working copy of your catalog is necessary to get it into shape for use with CVS. The following command creates a copy in the `/tmp` directory.

```
cp –a /var/lib/interchange/catalogs/foundation /tmp/import_foundation
cd /tmp/import_foundation
```

66.4. Streamline your catalog for CVS

66.4.1. Considerations about what to import into CVS

From your working directory (`/tmp/import_foundation`), decide what pages will be in the CVS repository, and which will not. While it is entirely possible to import the entire catalog into the repository unchanged, I usually prefer to doctor my directories up before letting them into my repository for several reasons:

- Will the file be modified by another source?
  For example, `/etc/order.number` is modified by interchange when run. But not everyone will use a local development model that includes running interchange on a directly checked−out copy of their source. Which means this specific issue is avoided if you upload every edit before viewing your changes on a server.
- The likelihood that you will modify the file.
  For example, if I am certain that I wont every want to modify the session/ files directly, then I probably wouldn't need to manage that through CVS, but I do import the empty session/ directory to make it easier when setting up new catalogs.
- Speed.
  Managing less files in the repository takes away from the amount of time required for cvs checkout, update, branching, and other cvs actions. For most, this amount of time is small already, but it is a consideration for some.
- Ease of use.
  One reason **NOT** to remove anything from your catalog before importing is it creates the ability to have a completely working catalog from just one checkout (much like the CVS tree at interchange.redhat.com). Whereas if you leave out other directories like etc/ session/ orders/, etc., then you must first combine your checkout with the other working parts of a catalog before the catalog is viable. But this is slower and will bring up lots of harmless notification and warning messages (about changed local versions) if you run interchange on your local source copy (because interchange will touch etc/ session/ orders/, etc. directly, and then warn that your local copy has changed from the CVS copy). You may be able to manage some of these notifications and warnings with `CVSROOT/cvsignore` or `$CVSIGNORE`, see the Resources appendix for more details.

66.4.2. Remove files that aren't needed in CVS

Here is an example of some directories to remove. If you do move more directories, be sure to move them to a directory that you can later use to re−unite with a checked−out copy for a working catalog. But here I chose just to move files that are not needed for a template "skeleton" catalog.
The images directory is typically symlinked to /var/www/html/foundation/images, so I remove this symlink from the working copy, and replace it with an exact copy which will go into the CVS repository.

```
cd /tmp/import_foundation
mkdir /tmp/import_foundation_nonCVS

#Setup images directory
rm images
cp −a /var/www/html/foundation/images .

#Remove
mv error.log logs/* orders/* session/* tmp/* upload/*  \
        /tmp/import_foundation_nonCVS
```

66.5. Import the streamlined catalog

Import the remaining portion of the catalog using the `cvs import` command, with "foundation" as the module name and repository directory name. See the CVS documentation resources mentioned in Appendix Resources for more information.
When you run the import command, it will launch $EDITOR (set to `'vi'` earlier), and ask for a message to go along with the import action. Whatever you see fit to write (e.g. "starting new cvs module with my foundation catalog...") is fine.
This example `import` command includes renaming the foundation "working" directory back to "foundation" for the import.

```
su - interch
cd /tmp/import_foundation
cvs import foundation foundation start
```

66.6. Testing the new CVS module

Now you should be able to do another test checkout or update using any CVS client, which should now download all the files that you have just imported into CVS. Additionally, you might test making a change to one of your checked-out source files, saving it, then committing it.

index.html:

```
<!--this is a test comment at the top of index.html-->
```

Now commit the change
`cvs commit index.html`
Your changed version will now be resident in the repository. There are a lot of good CVS documentation and resources for discovering more about the checkout/update/commit cycle and other CVS aspects in the Resources Appendix.
You'll also notice that even if you start your interchange server, the change you made did not take effect. The next section will detail the process of tying CVS and Interchange together in a way that this will happen automatically.

67. Integrate CVS and Interchange

The next step is to allow CVS to update the directory that Interchange uses to serve pages.

67.1. CVS checkout into the catalog directory

Now it is the time to replace the directories in your catalog that have counterparts in CVS with fresh checkouts from CVS (this is a preliminary action to allow CVS to update your catalog directory when a change is made to CVS).

Note: Make sure interchange daemon is stopped and you have a good backup before continuing.

`tar czf ~/foundation.backup2.tgz /var/lib/interchange/foundation`
Checkout a copy from CVS into a different directory (such as `foundation_CVS`).

```
cd /var/lib/interchange/
cvs co −d foundation_CVS foundation
```

This should create the `foundation_CVS/` directory for you, so that it wont conflict with your existing `foundation/` directory.

67.1.1. Add any needed files to checked−out catalog

If you removed any directories during the streamlining step, we must first add those back so that the catalog is usable to Interchange. In this document, we only removed files within the directories that weren't needed. This can also be the time to copy any "data" files such as orders/ logs/, etc. that might be needed if it is a live catalog.

```
cd /var/lib/interchange/foundation
cp −a <NEEDED_FILES> \
        /var/lib/interchange/foundation_CVS
```

67.1.2. Install and test the new catalog

Now lets move the old `foundation` out of the way and put the new `foundation_CVS` in its place.

```
cd /var/lib/interchange/
mv foundation foundation_old
mv foundation_CVS foundation
```

Now, link up the CVS images for use by Apache.

```
cd /var/www/html/foundation/
mv images images_old
ln −s /var/lib/interchange/foundation/images images
```

Now, you should have a working catalog again. To make sure, start up interchange and test the site with your browser.

67.2. Testing manual CVS updates on Interchange catalogs

Next, lets again update the checkout we made a while back before importing our catalog. (Alternatively, one could use a visual CVS client detailed above).

```
cd ~/src
cvs -q up -d foundation # -q for quiet, -d for directory prune/update
```

Additionally, you might test making a change to one of your checked-out source files, saving it, then committing it.

index.html:

```
<!--this is a test comment at the top of index.html-->
```

Now commit the change

```
cvs commit index.html
```

Your changed version will now be resident in the repository. Again, CVS documentation is in the Resources Appendix.
This time, we can allow the changes to take effect on the code being used by interchange to server pages. To do so, one must run a `cvs update` on the catalog directory:

```
cd /var/lib/interchange/foundation
cvs -q up -d    #up is the shortened version of "update"
```

That should notify you of the new version it downloaded with something like:
U pages/index.html
You may also get something like the following:

```
M catalog.cfg
M etc/status.foundation
M ...
? orders/000001
? ...
```

The `?` lines in the above example mean that the CVS server has never heard of the listed directories or files (they are in your local source dir but not in the CVS source dir). It is harmless, but sometimes annoying.
The `M` means that sthe file has been modified on your local copy, and is out of sync with the remote CVS version (e.g. when Interchange runs it updates `etc/status.foundation`). Normally this is corrected by uploading your "modified" version to the server, but in this case, the modification was done by Interchange instead of the programmer, and wasn't meant to be committed back to the CVS repository. These types of messages can be handled with `$CVSIGNORE` and `$CVSROOT/CVSROOT/cvsignore`.
Now, check to make sure that your change has taken effect by refreshing the homepage on the site. To see the comment, use `View->Page Source` or whatever the relevant command for your browser is.
At this point, its obvious that it would be time consuming to manually run 'cvs up' every time you make a change to the source code, so the next step is to setup CVS to automatically update the catalog whenever you commit something to CVS.

67.3. Automatic updates on commit

Start by modifying $CVSROOT/CVSROOT/loginfo

```
^foundation     (date; cat; (sleep 1; cd /var/lib/interchange/foundation; cvs -q update -d) &
```

The first line tells cvs that for every commit on modules that start with "foundation" (notice the regular expression `"^foundation"`), it will run `cvs update` on the given catalog directory in the background. It is important that it is executed in a forked shell (notice the `"&"`) after `sleep`'ing for 1 second, because otherwise you may run into contention issues that can cause file locking problems. The 1 second timing used above works fine for me, but a longer pause may be necessary for slower computers (you'll know if you get errors about "such−and−such locked by user so−and−so"). See the CVS documentation in the Resources Appendix for more details.

67.4. Automatic e−mail on commit

Often it is very helpful to have a commit mailing list that keeps developers up−to−date on every commit happening to the CVS. To setup automatic e−mails on every commit, put the following in

/rep/CVSROOT/loginfo:

```
ALL     /usr/bin/cvs-log     $CVSROOT/CVSROOT/commitlog $USER "%{sVv}"
```

This tells CVS to pipe the commit output to a shell script, which in turn updates a log file and e−mails an update (typically to a mailing list address). Create the shell script at `/usr/bin/cvs−log` that is executable by the cvs user (using `"chmod 755 /usr/bin/cvs−log"`).

/usr/bin/cvs−log:

```
#!/bin/sh
(echo "−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−";
 echo −n $2"  ";
 date;
 echo;
 cat) | tee $1 | /usr/bin/Mail −s "[foundation−cvs] $3" foundation−cvs@example.com
```

Your commit logs will now be archived in the CVSROOT/commitlog file, and e−mailed to the foundation−cvs@example.com address (of course, you would need to install a mailing list first). Here is what a sample e−mail looks like:

```
Subject: [foundation−cvs] 'directory/subdirectory filename.c,1.7,1.8'


−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
cvs Fri Mar 16 21:14:09 PST 2001
Update of directory/subdirectory
In directory cvs.foundationsomething.com:/tmp/cvs−serv7721
Modified Files:
filename.c
Log Message:
test
```

Now you have a working CVS development system. At this point it may be valuable to learn more about CVS the client tools that you are using.

68. The two track model: development and live catalogs

It is often very valuable to have a two−track development model that separates the classes of work into separate timing and decision categories. Some use "staging" and "production" terminology, others prefer "unstable" and "stable", "beta" and "release", or "development" and "live".
The easiest starting point for two−track development is to just use two completely separate CVS modules and catalogs. This can make a lot of sense for many situations, for example when the next revision of the site will be so different that it is almost starting from ground zero.
A slightly more complicated solution is to use the CVS branches feature. It is more difficult to set up, but can be rewarding when used correctly.

68.1. When to branch

The first decision is when to branch the source code. For websites, this can sometimes be an easy decision like "first went live", or "site−wide overhaul", etc.

68.2. Which way to branch

There are many different ways to branch source code. What seems to be the most common method is to use the HEAD (which is the default CVS mode of "no tag") as the development version, and then make a branch when a stable release is to be made.
That model doesn't fit my development style at the current time, so I use the HEAD default branch as my stable live version, and use other tags (like DEV1 and DEV_REALLY_UNSTABLE) for my development. You will probably find a method that fits your particular style as you learn more about CVS.
You will probably find that you are merging (or "folding") most or all of your development branch back into your stable branch frequently. This is because unlike traditional programming where products are launched every two or three years with new features, web sites often have little fixes and new features added every day or every few weeks, with new "releases" happening constantly (though not all web sites follow that trend). The flexibility is there to branch the source for quite some time to work on a very complex feature or complete redesign before bringing it to the live site.
Additionally, I prefer to not create a new branch every time I merge, though some do not mind the overhead.

68.3. Performing the branch

To perform the branch use the `cvs tag −b <BRANCH NAME>` command. For example:

```
cvs tag −b DEV1
```

Remember that this does not change your locally checked out working directory to the new tag automatically, it only creates the branch within the CVS repository.

68.4. Setup the development catalog

Now we have a branch in CVS, but we need to tie it to something in the real world, namely, an Interchange catalog.

68.4.1. Importing the catalog

Like we did in Integrating CVS with Interchange, you must make another copy of your catalog for use as the development version. Some would like to keep the orders/, logs/, and other directories the same, but I prefer to start with a clean slate, especially since I don't plan on having any customers visit the development site. (In fact, you can restrict who can access the development URL using the Apache `<Directory> allow from...` directive).

68.4.1.1. Checkout source code

```
cd /var/lib/interchange
cvs co −d foundation_dev foundation
```

68.4.1.2. Copy any other needed directories to complete the catalog

Depending on how complete your catalog is in CVS, you may need to create or copy directories/files.

```
cd /var/lib/interchange/foundation
cp −a catalog.cfg orders/*   \
        /var/lib/interchange/foundation_dev
```

Note: A lot of the following steps are performed by the /usr/local/interchange/bin/makecat script, but here is how to do it manually:

68.4.2. Setting up a separate database

Most often, I find it profitable to make use of a second database for the development catalog, rather than having both catalogs reference the same database (especially if the first catalog is live).

68.4.2.1. Create a second database

Use the means of your database platform to create a separate database. For example, PostgreSQL users might do something like:
`createdb foundation_dev`

68.4.2.2. Populate the database

You can rely on the catalogs internal products/*.txt data to generate the database tables and populate them, or you can export another catalog's database and import it for the development catalog, like the example below for PostgreSQL users.

```
pg_dump foundation  > ~/foundation.dump
psql foundation_dev < ~/foundation.dump
```

68.4.3. Copy the catalog support files

```
#Must be root
su − root

#Copy HTML
cd /var/www/html/
cp −a foundation foundation_dev
```

```
#Copy CGI
cd /var/www/cgi-bin
cp -a foundation foundation_dev
```

68.4.4. Configure the Interchange daemon

Many development catalogs will branch at the same time that they upgrade to a new Interchange daemon version. But for whatever interchange daemon version you use, perform the necessary modifications to `interchange.cfg`. For example:

```
/usr/local/interchange/interchange.cfg:
Catalog foundation     /var/lib/interchange/foundation     /cgi-bin/foundation
Catalog foundation_dev /var/lib/interchange/foundation_dev /cgi-bin/foundation_dev
```

68.4.5. Configure the catalog specifics

The development catalog will differ at least a little bit from the standard catalog, such as in the CGI_URL and database parameters. You can also modify/add the foundation_dev/variable.txt instead of the following.

```
/var/lib/interchange/catalog.cfg:
Variable CGI_URL   /cgi-bin/foundation_dev
Variable IMAGE_DIR /foundation_dev/images
Variable SQLDSN    dbi:Pg:dbname=foundation_dev
Variable SQLDB     foundation_dev
```

Now you can restart interchange to make your changes take effect.

68.5. Splitting updates on commit by tag

Setup CVS so that when you commit to the DEV1 branch, only the development (`foundation_dev`) catalog will be updated. And when you commit with no tags (HEAD branch), the live (`foundation`) catalog will be updated. Here is an example `loginfo`. Note the `-r DEV1` used in the cvs update command on the development catalog. This isn't strictly necessary, but it ensures that the right branch is used every time.

```
$CVSROOT/CVSROOT/loginfo:
^foundation      (date; cat; (sleep 1; cd /var/lib/interchange/foundation_dev; cvs -q up -d -
ALL     /usr/bin/cvs-log     $CVSROOT/CVSROOT/commitlog $USER "%{sVv}"
```

68.6. Using new branches

To use your new branch, checkout a working copy of the source with the correct tag specified. For example:

```
cvs co -P -r DEV1
```

Then make change to one of the files, and commit it. The change should show on your development catalog, but not your live catalog.

68.7. Merging

When you want to merge a change that you have made on your development branch into your stable

branch, there are many ways that you can do it. One would be to :

- Make a change in the development branch (DEV1) and commit it.
- Copy the development−tagged file to a temporary name
- Update to the live version (HEAD)
- Overwrite the live (HEAD) version of the file with your temporary one
- Commit the result
- Update back to the development version (DEV1)

I do the above so often that I have written a TCL script for WinCVS that will automatically perform the above steps. And similar shell scripts can probably be easily written to match your development environment.
The above seems to be the easiest way, to me. However, there are other alternatives detailed in the CVS manual in chapter 5, "Branching and merging", that I highly recommend for reading. One method involves specifying the last version that has already been merged into the live branch using a specific version number, date, relative time, or special purpose tag.

69. Tools of the trade

This is the productivity tips section, which will hopefully help you to be able to get more done in less time.

69.1. Workstation interchange installation

Not all developers work on Linux workstations, many use Apples (graphics designers / html gurus tend too, I've found), and many use Windows. This means that many developers have the extra step of uploading their changes to an FTP server or CVS server every time they make a change.
The remedy to that is to setup an interchange server on your workstation, or any location that has direct access to the CVS source files. I'll explain:
The interchange server that runs where the CVS server is (that we setup earlier) can be seen as the gathering point for all the developers. However, each developer may run as many interchange daemons as he/she requires in a local context for the purpose of seeing the changes made before uploading them via CVS.
For example, Bob could setup another interchange catalog on the same server as the CVS, (e.g. foundation−bob). To get direct access to those files (rather than FTP), Bob could use NFS mounts (if Bob's workstation is Linux) or SMB mounts using Samba if his workstation is a Windows variant. Any way that Bob can get direct access to the files will save him some time (by cutting out the "upload" from the "edit−>upload−>test" development cycle). One could even use Vmware to run a Linux server on your Windows workstation.

Note: You can now use the cygwin compatibility confirmed in later Interchange versions to run Interchange right on your Windows workstation.

The result will be that you can modify the files with your favorite text editor and see the results immediately through your local catalog. Setting up the catalog initially is quite easy. Just follow the same steps used to setup the CVS catalog. Which is:

- Stop interchange.
- bin/makecat a new catalog.
- Checkout from CVS into a new CVS catalog directory and link the images/ directory.
- Move any needed files back into the CVS catalog directory.
- Make modifications to products/variable.txt and catalog.cfg (e.g. CGI_URL, HOSTNAME, DBI_USER, DBI_PASSWORD).
- Restart interchange.

One aspect of this local configuration is managing the differences between the main interchange daemon which runs on the CVS server and the local interchange daemon. The differences are probably hostname, database information, etc. That will all need to be managed (usually through catalog.cfg entries) and database exports & imports (i.e. the postgres pg_dump command).
Another thing that you might have noticed at this point is all the files that are modified locally by the interchange daemon will report ? or M when you run an update. This can be handled with `CVSROOT/cvsignore` and `$CVSIGNORE`, which are beyond the scope of this document.

69.2. Mailserver for CVS updates

To setup a mailserver for CVS updates, first download and install mailman. For Red Hat systems, the following RPM could be used:

- ftp://speakeasy.rpmfind.net/linux/redhat/7.1/en/powertools/i386/RedHat/RPMS//mailman−2.0.1−2.i386.rpm

After installing, read the following information about Mailman and what needs to be done after installation (taken from the RPM meta data):
"Mailman is software to help manage email discussion lists, much like Majordomo and Smartmail. Unlike most similar products, Mailman gives each mailing list a web page, and allows users to subscribe, unsubscribe, etc. over the web. Even the list manager can administer his or her list entirely from the web. Mailman also integrates most things people want to do with mailing lists, including archiving, mail <−> news gateways, and so on.
When the package has finished installing, you will need to:

- Run `/var/mailman/bin/mmsitepass` to set the mailman administrator password.
- Edit `/var/mailman/Mailman/mm_cfg.py` to customize mailman's configuration for your site.
- Modify the sendmail configuration to ensure that it is running and accepting connections from the outside world (to ensure that it runs, set "DAEMON=yes" in /etc/sysconfig/sendmail, ensuring that it accepts connections from the outside world may require modifying /etc/mail/sendmail.mc and regenerating sendmail.cf), and
- Add these lines:

```
ScriptAlias /mailman/ /var/mailman/cgi-bin/
Alias /pipermail/ /var/mailman/archives/public/
<Directory /var/mailman/archives>
  Options +FollowSymlinks
</Directory>
```

to `/etc/httpd/conf/httpd.conf` to configure your web server.
Users upgrading from previous releases of this package may need to move their data or adjust the configuration files to point to the locations where their data is."
Then run `/var/mailman/bin/newlist` and follow the directions from there.

69.3. Locally mapped source code for a network IC server

This is useful mostly to Windows users, since Linux users can just as easily run IC daemons on their own workstation as they can a separate server.
The idea is to have the IC server use its own files and directories for things that won't be edited and modified locally, but reference a samba directory or NFS directory for things that will (such as `pages/`, `templates/`, etc.).

69.3.1. Mount the samba or NFS directory

`smbmount <...>` or `mount −t nfsfs <...>`
The following script uses two directories (source and destination) to create symlinks for the commonly modified source directories of Interchange.

```
export S=/mnt/nfs/foundation
export D=/var/lib/interchange/foundation
F=db; ln −s $S/$F $D/$F
F=dbconf; ln −s $S/$F $D/$F
F=etc; ln −s $S/$F $D/$F
F=images; ln −s $S/$F $D/$F
F=pages; ln −s $S/$F $D/$F
F=special_pages; ln −s $S/$F $D/$F
F=templates; ln −s $S/$F $D/$F
```

This will leave you with a working catalog that can be quickly modified (since your editor can access the local

copy), while interchange has to do the work of going over the SMB or NFS connection.

69.4. jEdit − a good editor with Interchange/HTML/perl colorization and CVS

I have been quite impressed with jEdit (http://www.jedit.org, and open source editor that is written in java and runs on most platforms.
I use the interchange.xml language definition written by Chris Jesseman chris@sitemajic.net, which is available from http://www.sitemajic.net/jedit/. With this, jEdit automatically colors HTML, perl, AND many interchange tags very intelligently.
Further, jEdit has a CVS plugin, written by Ben Sarsgard bsarsgard@vmtllc.com, and available at: http://www.vmtllc.com/~bsarsgard/jedit.html. This plugin allows you to diff, update, and commit right from the editor.

69.5. Seperate servers for development and live catalogs

If you have the luxury of seperate server ardware for the development and live catalogs, you might find the following utility helpful:

- CVSviaFTP (http://www.cvshome.org/dev/addoncvsftp.html) − from the CVS Add−ons page (http://www.cvshome.org/dev/addons.html).
  It allows one to have a given CVS module automatically publish each update to an FTP server, which could serve as the live server. Or one could could use it if your CVS installation is only local and you could use it to upload your changes to your production server.

G. Credits

- **Jon Jensen**: Thanks for helping me get going on the SDF format already used by the Interchange documentation, and fixing some SDF syntax errors.
- **Mike Heins & all who have contributed to the success of Interchange**: Thanks for following the Way Of The Source, for quality programming, and for helping to making IC something to write about.
- Thanks to the countless others who have written the CVS documentation that is available online, which was my only source for learning CVS.

H. Document history

- May 2001. Conceived and written by Dan Browning.
- July 19, 2001. First draft complete, first public release.

I. Resources

I.1. CVS Documentation

Here are some resources for learning more about CVS. I have ranked them by the order of usefulness, which is of course, objective.

- Karl Fogel's CVS book http://cvsbook.red−bean.com/
- The official CVS manual http://www.cvshome.org/docs/manual/
- The official CVS FAQ http://faq.cvshome.org/
- The official CVS homepage http://www.cvshome.org
- Info−CVS mailing list http://mail.gnu.org/mailman/listinfo/info−cvs
- CVS FAQ 2 http://www.cs.utah.edu/dept/old/texinfo/cvs/FAQ.txt
- Sean Dreilinger's CVS Version Control for Web Site Projects http://durak.org/cvswebsites/
- Pascal Molli's CVS reference site http://www.loria.fr/~molli/cvs−index.html
- CVS Tutorial http://cellworks.washington.edu/pub/docs/cvs/tutorial/cvs_tutorial_1.html
- CVS Tutorial 2 http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/cvs/
- RedHat CVS pserver setup guide http://www.michael−amorose.com/cvs/
- CVS Add−ons http://www.cvshome.org/dev/addons.html

I.2. CVS Server Software

- CVS RPM download (Red Hat 7.1)
  ftp://speakeasy.rpmfind.net/linux/redhat/7.1/en/os/i386/RedHat/RPMS/cvs−1.11−3.i386.rpm
- Source tarballs links can can be found at cvshome.org.

I.3. CVS Client Software

There are a variety of client access methods for using cvs on your development box.

- There are some great graphical clients for Linux, Windows, and Mac at http://www.cvsgui.org. These also give you the same access to all the command line cvs commands.
- jCVS is a great cross−platform graphical cvs client available at http://www.jcvs.org.
- jEdit is a great cross−platform text editor written in java, which not only has a CVS module that allows you to commit (upload) files directly from the editor, but also has a interchange markup language (and perl language) colorizer/parser. It is available from http://www.jedit.org.
