Interchange Back Office

Table of Contents

1. Interchange Back Office

Interchange is the industry's most widely distributed and implemented open source e−commerce platform. This document describes how to administer a commerce site with Interchange's back−office functionality, and discusses site management and security.

2. 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.

2.1. 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.

2.2. 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]
```

2.3. Custom 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 and `mv_order_route` form variable.

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  VEN  pgp_key         0x67798115
Route  VEN  email           orders@akopia.com
Route  VEN  reply           service@akopia.com
Route  VEN  encrypt         1
Route  VEN  encrypt_program "/usr/bin/pgpe -fat -q -r %s"
Route  VEN  report          etc/report_mail
```

This route would be used whenever the value `VEN` was contained in the form variable `mv_order_route`.

The last route that is defined provides the defaults for all other routes. For example, if `encrypt_program` is set there, then the same value will be the default for all routes.

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.

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.

credit_card

Determines whether credit card encryption should be done for this order. Either this or `encrypt` should always be set.

cybermode

If this is set, enables *CyberCash* for the route. Variables can also be set for `CYBER_CONFIGFILE`, `CYBER_SECRET`, and all other normal CYBERCASH variables. For example:

```
Route VEN cybermode         mauthonly
Route VEN CYBER_CONFIGFILE  config/vendor1_cfg
Route VEN CYBER_VERSION     3.2
```

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 `pgpe -fat -r %s`.

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`.

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
```

pgp_cc_key

The PGP 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`.

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`.

profile

The custom order profile which should be performed to check the order. 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.

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.

supplant

Whether this 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.

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 mode to change either `track` or `individual_track` files.

An 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.

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  encrypt_program  "/usr/bin/pgpe -fat -q -r %s"
Route  HARD  report           etc/report_mail

Route  SOFT  email            ""
Route  SOFT  profile          create_download_link
Route  SOFT  empty            1

Route  main  cybermode        mauthonly
Route  main  CYBER_VERSION    3.2
```

```
Route  main  CYBER_CONFIGFILE etc/cybercash.cfg
Route  main  pgp_key          0x67798115
Route  main  email            orders@akopia.com
Route  main  reply            service@akopia.com
Route  main  encrypt          1
Route  main  encrypt_program  "/usr/bin/pgpe -fat -q -r %s"
Route  main  report           etc/report_all
```

To tell Interchange that order routing is in effect, the variable `mv_order_route` is set on the final order submission form:

```
<INPUT TYPE="hidden" NAME="mv_order_route" VALUE="main">
```

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 arg=carts interpolate=1]
   my $string = <<'EOF';
[item-list][item-code]  [item-field goods_type]
[/item-list]
EOF
   my @items;
   my %route;
   @items = grep /\S/, split /\n+/, $string;
   for(@items) {
       my ($code, $keycode) = split /\t/, $_;
       $route{$code} = $keycode;
   }
   my $cart = $Carts->{'main'};
   my $item;
   foreach $item ( @{ $Carts->{'main'} } ) {
       $item->{mv_order_route} = $route{$item->{'code'}} || undef;
   }
   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).

The `main` order routing will use CyberCash to charge the order, and will be completely encrypted for e-mailing.

3. Administering Interchange

Some utilities are supplied with Interchange and are located in the VendRoot/bin directory:

```
compile_link Compiles an Interchange vlink or tlink CGI link
dump         Dumps the session file for a particular catalog
expire       Expires sessions for a particular catalog
expireall    Expires all catalogs
offline      Does offline build of the database(s)
update       Does in-place update of the database(s)
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 Internet Service Provider (ISP) that might have multiple users sharing an Interchange server. In particular, it is recommended that all of the session files and logs be put in a directory that is not writable by the user. If the directory or file is corrupted, the catalog may crash.

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

```
bin/interchange -test
```

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

3.1. Starting, Stopping, and Re-starting the Servers

The following commands need to have VENDROOT replaced with the main directory where Interchange is installed. If /usr/local/interchange is the site's Interchange base directory, the start command would be:

`/usr/local/interchange/bin/interchange`.

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

To start the server with default settings:

```
VENDROOT/bin/interchange
```

It is recommended to issue a restart, otherwise the server will not run anew if a server is already running.

```
VENDROOT/bin/interchange -restart
```

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.

To re-start the server:

```
VENDROOT/bin/interchange -restart
```

`-r` 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 server is running under. This information is also sent to /usr/local/interchange/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 /usr/local/interchange/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 case it will not stop, set:

```
VENDROOT/bin/interchange -kill
```

3.2. 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.

If only running a site with one method of communication, use the `-i` and `-u` flags.

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

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

3.3. 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`.

Use reconfigure from the command line (as the Interchange user) with:

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

It is easy to manually reconfigure a catalog as an administrator. 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.

3.4. Making the Product Database

The DBM product databases can be built off−line 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.

3.5. Updating Individual Records

If a site has a very large DBM database that takes time to build, consider using the `bin/update` script to change just one field in a record, or to add from a corrections list.

The following updates the products database `price` field for item 19−202 with the new value 25.00

```
update −c catalog −k 19−202 −f price 25.00
```

More than one field can be updated on a single command line.

```
update −c catalog −k 19−202 −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.

3.6. Expiring Sessions

If a site has DBM capability and Interchange is using it 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 /usr/local/interchange/bin/expireall −r
```

Interchange will wait until the current transaction is finished before expiring, so that 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, a Perl script can be used to delete all files not modified in the last one or two days. The following will work if given an argument of the 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 /usr/local/interchange/catalogs/construct/session
```

Give it multiple directory names, if there is more than one catalog.

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

3.7. Administrator Permissions

Select which operations each administrator can perform in the back office. Each section of the back office can be restricted with fine-grained control. An administrator can be given access to view the list of all orders, for instance, but not allowed to view details. Access to the rows of Interchange's internal tables can also be restricted on a per-table basis for each administrator.

3.8. Administrators

The Access Manager allows an administrator to create user accounts or groups of users and restrict the use of certain features. This feature is especially useful if a company has employees that need the ability to check orders, but not change Web content. Note that, by default, users in the back office are stored and managed separately from customer login accounts. Users can have permissions granted on an individual basis, or by group. If a user is a "super-user," all other permissions settings will be ignored and the user will be allowed to do anything.

3.9. Administrators: Edit Affiliates

Affiliates have the following attributes:

"Affiliate ID" is displayed in the order and traffic statistics along with the orders and traffic they produce.

"Affiliate Name" is the name of the affiliate.

"Campaigns" can be used to track traffic from advertising campaigns.

"Join_date" can be used to keep track of when the affiliate signed up.

"URL" is used, if present, to redirect visitors coming from this affiliate to a special home page just for visitors from that affiliate's site. This should not be the URL of the Affiliate's home site.

"Timeout delay" can be used to specify that orders attributed to this affiliate must happen within a certain amount of time from the time they were referred to the site by the affiliate. Measured in seconds.

3.10. Direct Table Edit

Edit any of Interchange's internal tables. Select a table to edit, or search a table for selected rows to edit.

3.11. Direct Table Edit: Select for Table Edit

Having selected a table to edit, a new row can be added, an existing row edited, all rows edited spreadsheet−style, or a row deleted.

3.12. File Transfer

Transfer pages, templates, and configuration files to and from the Interchange installation. Select `Pages` to transfer files that will be visible to site visitors. Select `catalog.cfg` to edit the configuration file for the store. `Upload` (send a file to the server), `download` (send a file from the server to a computer), `view`, or `edit` available files.

3.13. Import/Export

Interchange makes it easy to import and export data to and from a commerce Web site.

Use `Database Upload` to import a tab delimited database of all product information to Interchange to make set−up faster and easier. `Database Download` does just the opposite, allowing data to be downloaded from Interchange.

Use `Layout Upload` to upload a site's layout information. Use C>Layout Download> to download a site's layout information.

3.14. Logout

This feature will only be useful if there are multiple users in the Access Manager. When `logout` is clicked, a user will be asked to log in again. If `logout` is pressed in error, the user must log in again.

3.15. Meta Field Information

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.

4. Interchange Security

4.1. 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.

4.2. 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.

4.3. 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.

5. Usertag Reference

5.1. email

```
UserTag email Order to subject reply from extra
UserTag email hasEndTag
UserTag email Interpolate
UserTag email Routine <<EOR
sub {
    my($to, $subject, $reply, $from, $extra, $body) = @_;
    my($ok);

    $subject = '<no subject>' unless defined $subject && $subject;

    $reply = '' unless defined $reply;
    $reply = "Reply-to: $reply\n" if $reply;
    if (! $from) {
            $from = $Vend::Cfg->{MailOrderTo};
            $from =~ s/,.*//;
    }

    $extra =~ s/\s*$/\n/ if $extra;
    $ok = 0;
    SEND: {
        open(Vend::MAIL,"|$Vend::Cfg->{SendMailProgram} -t") or last SEND;
        print Vend::MAIL
                    "To: $to\n",
                    "From: $from\n",
                    $reply,
                    $extra || '',
                    "Subject: $subject\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" .
            "From '$from'\n" .
            "With extra headers '$extra'\n" .
            "With reply-to '$reply'\n" .
            "With subject '$subject'\n" .
            "And body:\n$body");
    }
    $ok;
}
EOR
```

5.2. email_raw

```
UserTag email_raw Documentation <<EOD

This tag takes a raw email message, *including headers*, and
users the SendmailProgram with -t option. 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.

EOD

UserTag email-raw hasEndTag
UserTag email-raw Interpolate
UserTag email-raw Routine <<EOR
sub {
    my($body) = @_;
    my($ok);
    $body =~ s/^\s+//;

    SEND: {
        open(Vend::MAIL,"|$Vend::Cfg->{SendMailProgram} -t") or last SEND;
        print Vend::MAIL $body
            or last SEND;
        close Vend::MAIL
            or last SEND;
        $ok = ($? == 0);
    }

    if (!$ok) {
        ::logError("Unable to send mail using $Vend::Cfg->{SendMailProgram}\n" .
            "Message follows:\n\n$body");
    }
    $ok;
}
EOR
```

5.3. loc

```
# [loc locale*] message [/loc]
#
# This tag is the equivalent of [L] ... [/L] localization, except
# it works with contained tags
#
UserTag loc hasEndTag   1
UserTag loc Interpolate 1
UserTag loc Order locale
UserTag loc Routine <<EOF
sub {
    my ($locale, $message) = @_;
    return $message unless $Vend::Cfg->{Locale};
    my $ref;
    if($locale) {
        return $message
            unless defined $Vend::Cfg->{Locale_repository}{$locale};
        $ref = $Vend::Cfg->{Locale_repository}{$locale}
    }
    else {
        $ref = $Vend::Cfg->{Locale};
    }
    return defined $ref->{$message} ? $ref->{$message} : $message;
```

```
}
EOF
```