Interchange Programmer Reference

Table of Contents

1. Introduction

Interchange is a highly−complex but very powerful web application server focused on ecommerce. It is built on the power of Perl, using many of its standard modules and capabilities while defining many more.

While Interchange focuses on e−commerce, it is really a general−purpose database access, retrieval, and templating systems. Besides online stores, here are some of the applications have been built on top of it:

```
Auction
Calendar
Configuration management
Content management
Document archival and rental
Guestbook
Image archival and download
Intranet
MP3 jukebox
Poll
Quiz
Software repository
Web log
```

This reference attempts to illuminate the source code of Interchagne and how you can write Perl enhancements, gadgets, and applications that integrate with Interchange.

1.1. Software installation

To follow along, it is recommended you get the latest release of Interchange (4.9.1 as of this writing), unpack it from the tar file, and install it at a private directory. For the purposes of this document, it will be assumed that Interchange is installed at `/usr/local/interchange` and that the catalogs are installed at `/usr/local/catalogs`.

1.2. Software prerequisites

Interchange only *requires* a few added Perl modules, which can be installed by getting the Perl CPAN bundle `Bundle::Interchange`. Install that (usually as root) with:

```
perl −MCPAN −e 'install Bundle::Interchange'
```

To get most of the modules Interchange can use:

```
perl −MCPAN −e 'install Bundle::InterchangeKitchenSink'
```

1.3. Audience

This reference is not meant for casual users of Interchange. Though they might learn something from reading it, it would probably not do them much targeted good. A reasonable set of prerequisites to make reading this document profitable include:

Programming knowledge

A good knowledge of Perl or **strong** knowledge of other programming languages is needed.

Database knowledge

Interchange is all about databases, and a knowledge of the concepts of database programming and SQL is strongly recommended.

Networking knowledge

The more you know about networking and the web, the more comfortable you will be with this document.

UNIX knowledge

Almost all production Interchange servers are UNIX−based, so knowledge of that is helpful.

2. Overview of Interchange

Interchange is a daemon server, similar to a web server. Its entry point is usually talking to it over a socket via its own protocol. That socket can be either UNIX domain or INET domain, or an infinite number of either.

2.1. Catalogs

Interchange as a server dispatches connections to a `catalog`, an independently−configurable set of data and templates. These are for the most part completely independent of each other, though they inherit common global characteristics and settings. Almost all of those can be overridden by the catalog.

2.2. Hacking

Of course Interchange's source is completely open and available. You could, if you wished, hack on it all you wanted. However this is strongly discouraged, for the simple reason that you can override almost any behavior with configurations and tag definitions of your own. In fact, if you want to override a core routine you can even do that.

So if you are tempted to hack a routine in the core, simply override it with:

```
GlobalSub <<EOR
sub override_me {
        package Vend::Interpolate;
        sub shipping {
                your_code();
        }
}
EOR
```

2.3. ITL −− Interchange Tag Language

Interchange delivers its content by parsing templates that contain text and ITL, tags in the Interchange Tag Language.

ITL takes the form of HTML−like tags using `[square brackets]` as the tag introduction. Here is an ITL tag sequence:

```
[if value name]
Your name is [value name], in case you forgot.
[/if]
```

The above will show the contents of the `[if ...] [/if]` container providing a non−blank, non−zero value is present in the user session.

ITL provides direct access to Perl via the ITL container tags `[perl]`, `[calc]` and `[calcn]`, and `[mvasp]`. This allows ITL like:

```
[calc]
        my $out = '';
        if($Values->{name}) {
```

```
                $out = "Your name is $Values->{name}, in case you forgot.";
        }
        return $out;
[/calc]
```

The above is completely identical to the ITL−only snippet above in effect.

In addition, you can call defined ITL tags in your embedded Perl:

```
[calc]
        my $out = '';
        my $name = $Tag->value('name');
        if($name) {
                $out = "Your name is $Values->{name}, in case you forgot.";
        }
        return $out;
[/calc]
```

Again, the result is identical to the previous two examples.

2.3.1. User Defined Tags

ITL is comprehensibly extensible. You can produce your own ITL tags that are fully as powerful as the ones supplied with the distribution. In fact they are indistinguishable, as you will see when you examine the `code` hierarchy.

These tags can use any Perl module, use external programs, or basically do most anything Perl can, providing you define them in the Global configuration. Tags defined in the Catalog configuration are restricted by Perl's standard Opcode and Safe facilities, though they can optionally be allowed global capability.

See the ictags manpage for complete information on ITL.

2.4. Talking to Interchange via socket

Interchange can run in any of several modes:

Foreground

The foreground, meaning the same Interchange server listens for connections and then runs the tasks those connections cause.

Forking mode

One master Interchange listens for connections, then forks instances to handle the tasks those connections cause. The forked instance terminates at the end of the task.

Prefork mode

Similar to the way Apache does, Interchange can fork off a number of instances that all listen to the sockets open for connections. The first one to answer gets the task, runs it, then returns to listen again. After MaxChildRequests requests, it dies and causes another new instance to take its place.

mod_perl mode

Interchange can be loaded into mod_perl. See the documentation in `scripts/ic_mod_perl.PL` for information.

SOAP mode

Interchange can listen to a socket designed to accept a SOAP connection −− those always run in prefork mode. This mode can co−exist with other modes, so the same Interchange server can serve both page and SOAP requests.

2.5. Talking to Interchange over the command line

Interchange starts its servers by being invoked from the command line. Other command line invocations can stop the server via signal, cause addition of additional catalogs to respond to, remove catalogs from the list to respond to, or cause execution of "cron" jobs.

2.6. Data structure overview

Interchange has three major data stuctures, which correspond to the master server, the catalog, and the user.

You can examine two of these structures by setting in interchange.cfg:

```
DumpStructure  Yes
```

This will by default dump an interchange.structure file which shows the global configuration, and a CATALOGNAME.structure file in each catalog directory showing that catalog's configuration.

The third structure, the user data session, can be viewed with the following ITL placed in a page:

```
<XMP>[dump]</XMP>
```

2.6.1. The Global configuration

This is held in a set of variables inhabiting the Global package. They define overall server behavior, and contain pointers to the catalog structures.

The Global configuration is defined in `interchange.cfg` and any files that it reads via `include` statements. The configuration is produced by parsing interchange.cfg with the routine `Vend::Config::global_config`.

Directives can be defined for parsing by the catalog configuration within the global configuration −− and they can be deleted as well.

The only way to define new global directives is via hacking the source. Luckily, this is just about never needed −− you can define settings for use by your programs in Variable or other repositories.

2.6.2. The Catalog configuration

Each Interchange catalog has its own configuration completely independent from others. It is basically produced from the file `catalog.cfg` in the directory defined as the base for the catalog. It is parsed by the subroutine `Vend::Config::config`.

We say basically, because there are many ways to alter catalog configuration. (CATNAME below refers to the name of the catalog being configured.)

ConfigAllBefore

Global catalog configuration preamble, affecting all catalogs, can be defined by the Global directive `ConfigAllBefore`. It defaults to `catalog_before.cfg` in the Interchange software directory (/usr/local/interchange).

LI1. CATNAME.before

An individual per−catalog preamble configuration is defined in `$Global::ConfDir/CATNAME.before`.

By default it would be /usr/local/interchange/etc/CATNAME.before.

CATNAME.site

A file in the catalog directory which is read before `catalog.cfg`. Deprecated.

catalog.cfg

The normal configuration file.

CATNAME.after

An individual per−catalog postamble configuration is defined in `$Global::ConfDir/CATALOGNAME.after`. This can be used to prevent user catalogs from doing unsafe things −− for instance enforcing the use of encryption, or preventing running in WideOpen mode.

By default it would be /usr/local/interchange/etc/CATALOGNAME.after.

ConfigAllAfter

Global catalog configuration postamble, affecting all catalogs, can be defined by the Global directive `ConfigAllAfter`. It defaults to `catalog_after.cfg` in the Interchange software directory (/usr/local/interchange).

command line

Any configuration passed on the command line at Interchange startup is applied last. For instance, to test out a catalog named `foundation` with a different invocation URL without having to alter the config files:

bin/interchange −−foundation:VendURL=http://localhost/cgi−bin/found \

```
◊ foundation:SecureURL=http://localhost/cgi−bin/found \
◊ foundation:RobotLimit=1000
```

That will set the `foundation` catalog directive values VendURL, SecureURL, and RobotLimit, overriding any settings in the configuration files.

Tied configuration

Interchange has dynamic catalog configuration as well. See Programming Watch Points in catalog.cfg.

2.7. Session data structure

Each user session is a hash reference saved in some sort of data repository. By default it is file−based using the Storable manpage, but it can reside in any Interchange database type as well.

It is placed at the global variable location $Vend::Session, which for programming use in UserTag and GlobalSub routines is $Session (meaning $Vend::Interpolate::Session).

The structure is initialized when the session is created (or canceled by the user). The initial form is described in `Vend::Session::init_session`:

```
$Vend::Session = {
            'ohost'         => $CGI::remote_addr,
            'arg'           => $Vend::Argument,
            'browser'       => $CGI::useragent,
            'referer'       => $CGI::referer,
            'scratch'       => { %{$Vend::Cfg->{ScratchDefault}} },
            'values'        => { %{$Vend::Cfg->{ValuesDefault}} },
            'carts'         => {main => []},
            'levies'        => {main => []},
};
```

This structure is used as a repository for the transitory user session values like form values, scratch variable settings, payment transaction results, errors, and any other user−tied values. It is also possible to add code that can be run on a user−by−user basis with the Autoload, Filter, and Profile facilities.

3. Tour the source

Navigating the Interchange source requires a couple of clues. The main program invocation point is `bin/interchange` in the Interchange software directory.

3.1. From startup to serving content

Once Interchange is invoked, it does some basic program configuration at the top of that file. The types of available database facilities and modules are determined, and the base modules are brought in with "use" or "require". Execution by a non−root user ID is checked.

After the initial program configuration, execution goes to the main_loop() subroutine in bin/interchange. Some more initialization is done, then the command line options are parsed. Options mostly will set the program mode (i.e. start, stop, kill, test, cron, or other command line actions), but can also set Global and Catalog configuration values.

Once the options are parsed, Interchange will chdir() to the Interchange software directory (/usr/local/interchange) and run its global configuration. That means all file names passed to it during this phase are relative to that program root.

Part of global configuration is determination of the ITL tags that will be used by Interchange. By default, that is all files with appropriate extensions under the `code` directory. Sets of tags to be used can be set with the TagGroup and TagInclude directives.

Global configuration also includes specifying the catalogs that will be configured and loaded in the next phase. This is done via the Catalog directive. An important part of that directive is supplying the `script` parameter, which is used to initialize the pointer structure which will select the catalog based on the URL coming in.

After Global configuration, catalog configuration commences, via the ::config_named_catalog() routine, which calls Vend::Config::config(). Each catalog specified in the global configuration has a base directory. Interchange does a chdir() to that directory and parses the various configuration files, databases and specified command−line parameters.

After the catalog is configured, the database is opened to ensure that database table objects are initialized properly. It is then immediately closed.

The resulting Catalog configuration structure reference is then saved in $Global::Selector and $Global::SelectorAlias so that the calling URL can map to the proper catalog.

Once all configuration is done, Interchange determines the program mode. There are only two modes −− `test` and `serve`. The test mode simply exits the program at this point −− it is used to test validity of the configuration.

If the mode is `serve`, ::main_loop() calls Vend::Server::run_server(). Based on global configuration, one of the server modes discussed previously is initialized and Interchange starts listening on one or more sockets for a connection from a client. (This is not true for mod_perl mode −− Interchange simply exits at that point and the code is waiting for mod_perl to call it.)

While waiting for a connection, signals are disabled and handlers are set up for TERM, HUP, INT, USR1, and USR2. TERM and INT both cause the main server to exit; HUP signals Interchange to look for a reconfiguration event; and USR1 and USR2 are optionally used to keep track of how many servers are running.

NOTE: Because signals are not especially safe in Perl prior to 5.8.0, occasionally a core dump can occur on receipt of USR1 or USR2. This is especially true for BSD with its reentrant system calls. They can be disabled by setting MaxServers to 0 −− PreFork mode is strongly suggested if that is done.

Once a connection is received, the connector parameters are checked for security constraints and Vend::Server::connection() is called. It reads the input from the client and constructs the environment, %CGI::values array, and any passed entity like an HTTP POST or multipart form (for file upload). Those are stored and and object referring to them and containing the connection file handle is constructed. That object is passed to main::dispatch() for processing.

The main::dispatch() routine performs more transaction setup then determines the catalog that will process the request. It sets $Vend::Cfg to the preset configuration for that catalog, sets file permissions as appropriate, and the catalog's database is opened.

Once initialiation of the catalog configuraion is complete, user initialization begins. Interchange determines the user session ID, if any, and restores the user session from the session database or starts a new session as appropriate. Perl objects that will be used in the session are initialized or constructed, auto−login is run, and the locale is determined and set. After that, the URI path is parsed, Autoload and Filter routines are run.

Finally a transaction action is determined. The action is the first path component of the path passed to Interchange. The remainder is passed to the subroutine implementing the action, and may be used as default path information for content or for other purposes.

For example, if the catalog VendURL is `/cgi−bin/foundation` and the URI sent to Interchange is `/cgi−bin/foundation/order/something/or/another`, the action is `order`, and the path sent to the action routine is is `something/or/another`.

If the transaction action is not mapped via standard system actions defined in the variable %action, or in the ActionMap *global* or ActionMap directives, then the action path component is restored to the content path, and that page is served (`order/something/or/another` in the example above).

If the action is mapped, it is run. If it returns a true value, the page to be served is determined by the setting of $CGI::values::mv_nextpage. The action can produce send its own output and return a non−true value, in which case Interchange will terminate the transaction at that time.

After the action is run and/or content is served, Interchange runs AutoEnd, saves the user session, closes the catalog database, and finally main::dispatch() returns. The calling Vend::Server::connection() does some cleanup and returns to the server loop. If the server was forked for that transaction only, it sends a signal indicating it is done, cleans up PID files, and exits. If it is in the foreground or in PreFork mode, it scrubs the Vend:: and CGI:: namespaces and returns to waiting for the next connection.

3.2. Notes about databases

Interchange maintains objects for all of its database tables defined in Database. These can be of diverse SQL, DBM, and LDAP types.

When the database is initialized at catalog configuration time, the individual database tables may be opened depending on type. In general, SQL and LDAP types are always opened, and DBM types are not.

Opening a database table can be expensive in terms of CPU and IO time. So when the database is opened for a page transaction, Interchange creates a "dummy" table object that waits for a real access. Those objects are trivial to create, and a fast processor can create hundreds of thousands per second.

When access is made, the database table is really opened and the expensive initialization is done. This allows many tables to be ready for access while only the ones used take up CPU and IO time.

4. Interchange Special Variables

Interchange defines some special variables which control behavior. They can be of several types, and the conventions for using them depend on whether you have based your catalog and server on the standard "foundation" distribution.

We will distinguish between these by calling intrinsic variables CORE variables, noting the distribution variables as DISTRIBUTION, and noting the foundation catalog practices as FOUNDATION.

4.1. "Variable" configuration file definitions

Defined in interchange.cfg or catalog.cfg with the `Variable` configuration directive, these are accessed with:

```
Access in ITL with           From
----------------------       -------------------
__VARNAME__                  (catalog.cfg only)
@_VARNAME_@                  (catalog.cfg, falls back to interchange.cfg)
@@VARNAME@@                  (interchange.cfg only)
[var VARNAME]                (catalog.cfg only)
[var VARNAME 1]              (interchange.cfg only)
[var VARNAME 2]              (catalog.cfg, falls back to interchange.cfg)

Embedded Perl                From
----------------------       -------------------
$Variable->{VARNAME}         (catalog.cfg only)
$Tag->var('VARNAME')         (catalog.cfg only)
$Tag->var('VARNAME', 1)      (interchange.cfg only)
$Tag->var('VARNAME', 2)      (catalog.cfg, falls back to interchange.cfg)
$Global::Variable->{VARNAME} (interchange.cfg only, only in Global code)
```

Variables set with `Variable` are not normally modified dynamically, though you can do it as a part of the `Autoload` routine or in other code. They will not retain the value unless `DynamicVariables` is in use.

4.2. Scratch

User scratch variables are initialized whenever a new user session is created. They start with whatever is defined in the `ScratchDefault` directive in catalog.cfg; otherwise they are not defined.

```
Access in ITL with           Attributes
----------------------       -------------------
[scratch varname]            Displays
[scratchd varname]           Displays and deletes

Embedded Perl                From
----------------------       -------------------
$Scratch->{varname}          Accessor
$Session->{scratch}{varname} Equivalent
```

They can be set in several ways:

```
Set in ITL with              Attributes
----------------------       -------------------
[set varname]VAL[/set]       Sets to VAL, no interpretation of ITL inside
```

```
[seti varname]VAL[/seti]     Sets to VAL, interprets ITL inside
[tmpn varname]VAL[/tmpn]     Sets to VAL, no ITL interpretation, temporary
[tmp  varname]VAL[/tmp]      Sets to VAL, interprets ITL inside, temporary

Embedded Perl                From
----------------------       -------------------
$Scratch->{varname} = 'VAL'; Sets to VAL
$Tag->tmp(varname);          Set as temporary, must set value afterwards.
```

4.3. CGI

CGI variables are the raw data which comes from the user.

WARNING: It is a security risk to use these variables for display in the page.

You can use them for testing without worry, though you should never set their value into a database or display on the page unless you have processed them first, as they can have arbitrary values. The most common security risk is displaying HTML code, which allows remote scripting exploits like cookie-stealing.

```
[calc]
    ####  DO NOT DO THIS!!!!
    my $out = $CGI->{varname};
    return $out;
[/calc]
```

That will transform the value. If you wish to output a safe value but keep the actual value intact, do:

```
[calc]
    ####  This is safe, makes value safe for rest of page
    my $out = $Tag->cgi( { name => 'varname', filter => 'entities' } );
    ####  This is safe too, doesn't transform value
    my $other = $Tag->filter($CGI->{varname}, 'entities');

    ### Now you can return stuff to the page
    return $out . $other;
[/calc]
```

The access methods are:

```
Access in ITL with                Attributes
-----------------------           -------------------
[cgi varname]                     Doesn't stop ITL code, don't use!
[cgi name=varname filter=entities] Use this for safety

Embedded Perl                From
----------------------       -------------------
$CGI->{varname}              Don't use for output values!
```

They can be set as well.

```
Set in ITL with                      Attributes
-----------------------              -------------------
[cgi name=varname set="VAL"]         Sets to VAL, VAL can be ITL, shows VAL
[cgi name=varname set="VAL" hide=1]  Sets to VAL, VAL can be ITL, no output

Embedded Perl                From
```

```
----------------------      ------------------
$CGI->{varname} = 'VAL';    Sets to VAL, next access to [cgi varname]
                            shows new value
```

4.4. Values

User form variables are initialized whenever a new user session is created. They start with whatever is defined in the `ValuesDefault` directive in catalog.cfg; otherwise they are not defined except as called out in other configuration directives, i.e. the obsolete DefaultShipping.

```
Access in ITL with          Attributes
----------------------      ------------------
[value varname]             Displays

Embedded Perl               From
----------------------      ------------------
$Values->{varname}          Accessor
```

They can be set as well, though the normal method of setting is from user input via form. If Interchange receives an action which performs the update of values (by default `go` or `return`, `refresh`, or `submit`), the value of CGI variables will be transferred to them subject to other considerations (FormIgnore settings, credit card variables, etc., discussed below).!block example

```
Set in ITL with                         Attributes
----------------------                  -------------------
[value name=varname set="VAL"]          Sets to VAL, VAL can be ITL, shows VAL
[value name=varname set="VAL" hide=1]   Sets to VAL, VAL can be ITL, no output

Embedded Perl                           Attributes
----------------------                  -------------------
$Values->{varname} = 'VAL';             Sets to VAL, next access to
                                        [value varname] shows new value
```

4.5. Session variables

You can also directly access the user session. Normally you don't set these values unless you are an experienced Interchange programmer, but there are several values that are frequently used.

One example is `username`, which holds the logged-in user's username.

```
Access in ITL with          Attributes
----------------------      ------------------
[data session username]     Displays

Embedded Perl               From
----------------------      ------------------
$Session->{username}        Accessor
```

They can be set as well, but if you are experienced enough to contemplate doing these things you will easily be able to figure it out.

4.6. Values not transmitted from CGI

The following variables are never copied from CGI:

```
mv_todo
mv_todo.submit.x
mv_todo.submit.y
mv_todo.return.x
mv_todo.return.y
mv_todo.checkout.x
mv_todo.checkout.y
mv_todo.todo.x
mv_todo.todo.y
mv_todo.map
mv_doit
mv_check
mv_click
mv_nextpage
mv_failpage
mv_successpage
mv_more_ip
mv_credit_card_number
mv_credit_card_cvv2
```

You can define more with the `FormIgnore` catalog.cfg directive.

4.7. Global program variables

If you are programming a GlobalSub or global UserTag, you have access to all Interchange facilities including all the preset variables and configuration directives.

The `Global` package is used to hold variables that are set at program start and whose value is retained.

The `Vend` package is used for variables that might be set at some point during program execution, but that will always be reset to undefined at the end of the transaction.

One example is $Vend::Cookie, which holds the raw cookie value sent by the user.

If you are going to set or access these variables, you should be getting your documentation from the source code. A few will be shown here, but most will not.

5. Variable listing

5.1. Standard global (interchange.cfg) Variable values

Copyright 2002 ICDEVGROUP. Freely redistributable under terms of the GNU General Public License.