

# **Interchange: Catalog–Building Tutorial**



# Table of Contents

<b><u>1. Purpose</u></b> .....	<b>1</b>
<b><u>2. Preliminary</u></b> .....	<b>3</b>
<u>2.1. Install Interchange and the demo catalog</u> .....	3
<u>2.2. Why we need the demo catalog</u> .....	3
<u>2.3. The Interchange operating system user</u> .....	3
<u>2.4. Important directories</u> .....	3
<u>2.5. Your catalog URL</u> .....	4
<u>2.6. Starting or restarting Interchange</u> .....	4
<u>2.7. Our assumptions</u> .....	5
<b><u>3. Starting our catalog</u></b> .....	<b>7</b>
<u>3.1. Create the link program</u> .....	7
<u>3.2. Create the tutorial catalog directory</u> .....	7
<u>3.3. Become the Interchange user</u> .....	7
<u>3.4. Go to the tutorial catalog directory</u> .....	7
<u>3.5. Create the session directory</u> .....	8
<b><u>4. Configuration files</u></b> .....	<b>9</b>
<u>4.1. interchange.cfg</u> .....	9
<u>4.2. catalog.cfg</u> .....	9
<b><u>5. The products database table</u></b> .....	<b>11</b>
<u>5.1. products/products.txt</u> .....	11
<b><u>6. Page templates</u></b> .....	<b>13</b>
<u>6.1. top</u> .....	13
<u>6.2. left</u> .....	13
<u>6.3. bottom</u> .....	13
<u>6.4. Interchange Tag Language</u> .....	14
<b><u>7. A welcome page</u></b> .....	<b>15</b>
<u>7.1. pages/index.html</u> .....	15
<b><u>8. Troubleshooting</u></b> .....	<b>17</b>
<b><u>9. Displaying products</u></b> .....	<b>19</b>
<u>9.1. Listing all products</u> .....	19
<u>9.2. pages/flypage.html</u> .....	20
<b><u>10. Unknown URL error page</u></b> .....	<b>21</b>
<u>10.1. special_pages/missing.html</u> .....	21
<b><u>11. The shopping basket</u></b> .....	<b>23</b>
<u>11.1. A link for ordering</u> .....	23
<u>11.2. pages/ord/basket.html</u> .....	23

# Table of Contents

<b><u>12. Order checkout</u></b> .....	<b>25</b>
<u>12.1. pages/checkout.html</u> .....	25
<u>12.2. etc/profiles.order</u> .....	27
<u>12.3. special_pages/needfield.html</u> .....	27
<u>12.4. Credit card processing</u> .....	28
<u>12.5. etc/report</u> .....	28
<u>12.6. special_pages/receipt.html</u> .....	29
<b><u>13. Enhancing the catalog</u></b> .....	<b>31</b>
<u>13.1. Price pictures</u> .....	31
<u>13.2. Catalog variables</u> .....	32
<u>13.3. A slightly more interesting page footer</u> .....	32
<u>13.4. Fancy credit card expiration date selection</u> .....	33
<u>13.5. Sorting the product list</u> .....	34
<u>13.6. A search box</u> .....	35
<u>13.7. Default catalog page</u> .....	36
<u>13.8. High–traffic changes</u> .....	37
<b><u>14. Ideas for further enhancements</u></b> .....	<b>39</b>
<u>A. Catalog directory structure</u> .....	<b>41</b>
<u>B. Document history</u> .....	<b>43</b>

# 1. Purpose

This document is a tutorial that guides you through constructing a simple Interchange catalog. The demo catalogs that ship with Interchange have become rather complex since they attempt to highlight some of the many capabilities of Interchange offers. As a template for your own catalog, the demos can be a somewhat intimidating place to start.

The simple catalog you create in this tutorial should give you a feel for the Interchange system. It should also be a stepping stone to a more complete and functional ecommerce system built with Interchange. To this end, this tutorial will take a minimum of initiative and rely as much as possible on default settings. It will use as few of Interchange's capabilities as possible, while still building a usable store. The resultant site will be simple. It will be unappealing. Your friends would make fun of you if you used it in real life. It should, however, be instructive.

We encourage you to create the files used in this tutorial yourself. Compared to just downloading ready-to-go files, you'll learn much more when you create the directory structure and use your favorite text editor to create files in the proper places on your own system as they are discussed.



## 2. Preliminary

### 2.1. Install Interchange and the demo catalog

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

Detailed installation instructions are beyond the scope of this document. However, you need to know which type of installation you have so you know where to place the various files we will create. Before going any further, verify that Interchange is properly installed and the demo catalog runs as expected, and note which type of installation you have:

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

Huge amounts of frustration can be avoided by making sure that the demo works correctly before venturing off on your own.

### 2.2. Why we need the demo catalog

In this tutorial, we want only to create a single installed catalog. By contrast, we normally discuss catalog **templates** such as *construct*, *barry*, and *simple*, from which one can create multiple new catalogs, set up in various ways. The `makecat` program is used to generate individual catalogs from catalog templates, but we want to avoid using it for this tutorial to keep things simple. So, we're going to take a few shortcuts that rely on the demo catalog having already been installed by `makecat`.

Installing the demo catalog did a few things for us. First, it set up the Interchange global configuration file `interchange.cfg`, which resides in the Interchange software directory. Second, it compiled the link program for your specific server and placed the executable program in your `cgi-bin` directory.

### 2.3. The Interchange operating system user

If Interchange was installed as a regular user, that will be the user Interchange will run as. But if Interchange was installed as root or from an RPM, you need to know the name of the separate Interchange user. The Interchange daemon does not run as root, or even as the web server user (often `www` or `httpd` or `nobody`), but rather as its own user. Usually the username is `interch`. If Interchange was installed from RPM, or with the default source installation settings, this will be the case. If a different user name was chosen, you will need to find out what it is.

### 2.4. 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:** Installation of Interchange is extremely flexible and the exact locations on your system may vary. Please do yourself a favor and do not proceed until you are sure you have this information and have the necessary permissions to write to these directories.

---

## 2.5. Your catalog URL

Finally, you need to know the URL to access your store from a web browser. Again, this can vary widely depending on the setup of your web server. But given a common setup of the Apache web server, it is likely to be:

- 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 itself, you'll need to substitute your server's host name (something like `machine.domain.com`) for `localhost` whenever we mention URLs.

## 2.6. Starting or restarting Interchange

You may have the Interchange server running already, but whenever you change the configuration files you will need to restart the Interchange server. How this is done depends (like so many other things) 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, as you'll need to restart Interchange a few times during the tutorial.

## 2.7. Our assumptions

For the sake of simplicity, **the remainder of this tutorial assumes you've installed Interchange under Red Hat 7 from the RPM onto your own workstation.** This gives us 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: construct
- Demo catalog URL base: http://localhost/cgi–bin/construct
- Tutorial catalog name: tutorial
- Tutorial catalog URL base: http://localhost/cgi–bin/tutorial
- Tutorial catalog directory: /var/lib/interchange/tutorial

Substitute the correct values for your system when we mention these settings in the tutorial.

---

**Note:** Unless you are fairly experienced with Unix, your web server, and the basic layout of the Interchange software, we highly recommend using the RPM installation with Red Hat Linux or Linux Mandrake.

---



## 3. Starting our catalog

### 3.1. Create the link program

We will now 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 `construct`. This file links the Interchange daemon with your web server. You need to make sure that it has the same owner and file permissions as the one you copied from. Especially important is the set-UID bit (unless you installed as a regular user). You will probably need to be root to have write permission in the `cgi-bin` directory.

Typing this command as root while in your `cgi-bin` directory should do what you need:

```
cp -p construct tutorial
```

If everything's working correctly, typing `ls -l construct tutorial` should describe your files roughly like this:

```
-rwsr-xr-x  1 interch  interch      7708 Dec 16 22:47 construct
-rwsr-xr-x  1 interch  interch      7708 Dec 16 22:47 tutorial
```

### 3.2. Create the tutorial catalog directory

Create a subdirectory named `tutorial` under your catalogs directory (probably `/var/lib/interchange/`). This is where all of our catalog-specific files will go. It needs to be readable, writable, and executable by the Interchange user. We will refer to this as your *catalog directory* (singular, not plural). These commands, run while in the catalogs directory, should do what you need:

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

### 3.3. Become the Interchange user

From here on out, you should be able to do everything you need to do as the 'interch' user. So do `su interch` now. You'll have to supply user `interch`'s password if you weren't root. (If you installed Interchange from RPM, user `interch` probably doesn't have a password. You'll have to set it with the `passwd` command while root.)

### 3.4. Go to the tutorial catalog directory

Change to the catalog directory with the `'cd'` command now. **From here on out, 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 and is described below.

Note that to improve clarity, we will append a trailing slash to directory names, to more clearly distinguish

them from file names. (This is similar to the output of the `ls` command with the `-F` option.)

### **3.5. Create the session directory**

You need to create the session directory, which is where Interchange will save information on each visitor's browsing session. If you do not have this directory, Interchange may fail to work for you in what appears to be a mysterious way. This directory is called `session/` and goes under your catalog directory. A simple `mkdir session` should do the trick.

## 4. Configuration files

Interchange configuration is controlled by a number of directives, which are specified in two kinds of configuration 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. It doesn't matter how many spaces or tabs are between the directive and its options, but the directive and all its options must be on the same line (with a few exceptions that don't concern us here). The case of the directive isn't important, but it's recommended that you preserve the capitalization given for readability and consistency.

You are free to insert blank lines or comment lines (lines where the first non-blank character is '#') freely throughout the configuration files to improve readability. The order the lines appear in is significant, but won't matter for our simple catalog.

Now it's time to pull out your favorite text editor (`vi`, `emacs`, `pico`, `joe`, `gedit`, and `nedit` are all favorites) and start editing some files.

### 4.1. interchange.cfg

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

```
Catalog name catalog_base_directory link_url_path
```

Open `interchange.cfg` (remember, it is in the Interchange software directory). Go near the top of the file, right below the other `Catalog` directives. Add this line:

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

That's it for this file! Save it.

### 4.2. catalog.cfg

From here on out, most of the files we mention don't exist yet. You'll create them yourself with initial text we give.

We need to create a `catalog.cfg` file for our tutorial store (back 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 recognizes a number of file formats. We will use a tab-delimited text file. Enter this line into `catalog.cfg`:

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

## Interchange: Catalog–Building Tutorial

This tells Interchange that we have a database table named 'products' which 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. Since Interchange is at heart an electronic commerce system, it expects to find at least a database table of products. We can specify all of the database tables that contain products by using the ***ProductFiles*** directive. There is no default for this, so we will have to specify our products database by adding the following line to `catalog.cfg`:

```
ProductFiles products
```

There are a few other directives which Interchange expects to see in order to complete our minimum configuration. These are ***VendURL***, ***SecureURL***, and ***MailOrderTo***. These are, respectively, your catalog's base URL, its secure URL, and the e–mail address to mail order notices to. We will supply these by adding the following lines to `catalog.cfg`:

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

Your first brand–new file, `catalog.cfg`, 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
```

## 5. The products database table

### 5.1. products/products.txt

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

The `products/products.txt` file will serve two purposes. It will provide Interchange with the layout of the products database table and it will 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. The first field in the list is expected to be a primary key (unique identifier) for that row. In this case we 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 by Interchange. It expects to find at least a description and a price field in addition to the product ID. In other words, this file must at least contain fields named "sku", "price", and "description". Any other fields you wish to include are handled normally.

Since this is kind of a test catalog, let's sell tests. Digging through my files, I find the following tests from classes I've taken. 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, and spaces or anything else will mess things up.

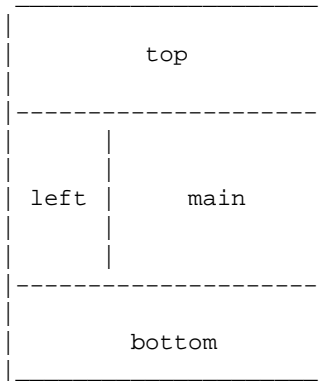
---

You may notice (or you may not) that the columns don't line up in your text editor. If you are tempted to fix this to make it easier to read, maybe you missed my warning.



## 6. Page templates

Now we need something to display. Let's start by assuming that certain aspects of our site will remain the same as the content of the different pages changes. We are going to create a template that we can use for all of our pages. We'll divide our pages into four sections like so:



The main section holds the content which will be different for each page. The top is for headers, banners, menus and the like. The left could be used for a sidebar or something similar, and the bottom can be the copyright and contact info. The top, left, and bottom sections will remain constant throughout the site.

Now create the HTML for each template section in its own plain text file in the catalog directory, named simply 'top', 'left', and 'bottom', respectively. Why no '.html' suffix on these? Mostly to make clear that they're not meant to be parsed directly by Interchange as full pages. Why no '.txt' suffix? To make our [include] tags simpler (as you'll see below). And because we can.

### 6.1. 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>
```

### 6.2. left

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

### 6.3. bottom

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

```
</body>  
</html>
```

Since this isn't an HTML tutorial, there will be no commentary on these fragments.

### 6.4. Interchange Tag Language

Now we need a way to pull the template pieces into the proper places on our pages to make a complete page. It's time to introduce you to ITL, the Interchange Tag Language.

ITL is at the heart of almost all Interchange catalog pages. It's how you tap Interchange's functionality. The tags spoken of appear between square brackets like [this]. Options appear after the tag, separated by whitespace, like this: [tag value1 value2] and this: [tag option1=value1 option2=value2]. They may span multiple lines. (That can help readability when the tag has many options.) There are many tags, and we will only scratch the surface of what can be done. Please refer to the ITL Tag Reference for full details.

Our 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. We'll see how this works on the page we need to create next.

## 7. A welcome page

### 7.1. pages/index.html

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

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

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

Let's see if it works. Restart Interchange so your changes take effect. Now go to your web browser and load the page. The URL should look like `http://localhost/cgi-bin/tutorial/index.html`.

---

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

---



## 8. Troubleshooting

Hopefully your first Interchange page worked well. If it didn't, you'll need to try to figure out what went wrong. The vast majority of the time, overlooked details is the problem. Double-checking your typing is a good habit to get into.

Here is a troubleshooting checklist that will come in handy whenever 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 our tutorial catalog.)
2. Have you misspelled any file names or put them in the wrong directories? 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 type whitespace correctly in the cases where it mattered? Remember to use tabs when tabs are called for (in lists and database text files).
6. Check your catalog error log, `error.log` in your tutorial catalog directory, to see if Interchange reported anything of interest.
7. Check the Interchange server error log, `error.log` in the Interchange software directory, to see if it had problems loading the catalog at all.
8. View the HTML source of any catalog pages that are messed up. The problem may reveal itself when you see what HTML the browser is getting.



## 9. Displaying products

### 9.1. Listing all products

Now that we have the store running let's display our 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 ..." 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 our products database. The `[loop]` `[/loop]` pair tells Interchange to iterate over each item provided 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 code should take the place of the above ellipsis in `index.html`:

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

Inside the loop we access the individual elements of the entry using the `[loop-field]` tag. The following code should replace the above ellipsis 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. Here it will produce the same output as `[loop-field sku]`, because the 'sku' field is the primary key for table products.

In each case the tag is replaced by the appropriate element. Put it all together and Interchange will generate 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]

```

Now is a good time to test this. Go ahead and refresh the `index.html` page in your browser.

## 9.2. pages/flypage.html

Now let's go ahead and create an individual page for each item. All we have to do is 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 we request `0198.html`, Interchange will first check for a page with that name. When it doesn't find one, it will search the products database table and see there is a product with that ID. Then it will create a product page "on the fly" using `pages/flypage.html`. When constructing the flypage, the entire product record for the requested product is available via the `[item-field]` tag (much like the `[loop-field]` tag). So, let's create `pages/flypage.html` like so:

```

[include top]
[include left]

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

[include bottom]

```

To provide links to the product flypages from our home page we will modify `pages/index.html` slightly, so that:

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

becomes:

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

# 10. Unknown URL error page

## 10.1. special\_pages/missing.html

Create the `special_pages/` directory in your tutorial catalog directory now. (**Not** in the `pages/` directory!)

We mentioned above that Interchange wants to display an error page when it is asked for an unknown page. Let's create `special_pages/missing.html` for this purpose. Make it look like this:

```
[include top]
[include left]
<p>Page not found, good buddy.</p>

<p>[page index]Go to welcome page[/page]</p>
[include bottom]
```

That will ensure users see a mildly friendly message instead of a mysterious server error if they mistype a URL.



# 11. The shopping basket

## 11.1. A link for ordering

Now that we have the products available for perusal, let's add a shopping cart so that folks can actually purchase our wares. For this we will use the `[order]` `[/order]` tags. These create an HTML link which will cause the specified item to be ordered and then will transfer the shopper to the basket page. This is actually just 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 to add is marked by a '+'. Do not include the '+'. The other lines are shown to give you context. (This style is called a "context diff" and will be used frequently.)

---

## 11.2. pages/ord/basket.html

Create the directory `pages/ord/` inside the tutorial catalog directory now. In other words, you should have an `ord/` *inside* the `pages/` directory.

For the `[order]` tag Interchange will expect a default page called `pages/ord/basket.html`. This page will display the contents of the shopping basket as well as have other shopping basket functionality.

The Interchange demo store has a full-featured shopping basket available for you to use, but we'll cobble our own simple one together. The shopping basket items can be accessed using a set of tags which have an `[item]` prefix. Let's put this text in the new file `pages/ord/basket.html` first and then we'll discuss the tags used in it:

```
[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[/page]<br>
[page index]Return to shopping[/page]
</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] will show the quantity of this item ordered. If the same item is ordered multiple times, this quantity will increase.
- [item–field description] shows the *description* from the product database table. Any field which is not special to Interchange can be accessed from the shopping cart this way.
- [item–price] is the per–item price which is defined in the product database table.
- [item–subtotal] is the total cost of this order line. This is nominally price multiplied by quantity but can also take into account other considerations, such as various kinds of price discounts.
- [subtotal] is the calculated shopping basket subtotal.
- [page index]...[/page] creates a link to the catalog welcome page.

Now we will also 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` thusly:

```

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

```

Go ahead and test out the shopping basket in your browser.

# 12. Order checkout

## 12.1. pages/checkout.html

We can now complete the site by adding the ability to check out with the shopping cart and finalize the order. To do this we want the customer to provide a shipping address (which we will lazily assume is the same as the billing address), and payment information. We will process the order by verifying their payment information and sending an email to the merchant (ourselves) detailing the order.

We'll start by creating a checkout page. The checkout page will consist of a form that takes the pertinent information from the customer and performs a simple credit card number check. In this case we'll use the built-in test that only checks to see if a given card number could be valid. If all of the information is acceptable the customer will move on to the next phase of the order process. If it is not, an error page will be displayed.

Let's create `pages/checkout.html` and then we can discuss it step by step. This page is a bit more complex than the others, but don't be intimidated. Most of it is just standard HTML form coding.

```
[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>
```

## Interchange: Catalog–Building Tutorial

```
</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[/page]</p>
[include bottom]
```

First we begin an HTML form 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 which is configured by submitting certain fields in the form. The Finalize button will invoke this form processor and then link the user to the `special_pages/receipt.html` page (described later).

We will submit some hidden form values which will tell Interchange how to process this form. The first value *mv\_todo* has the value of *submit*; this causes the form to be submitted for validation. The second value sets *mv\_order\_profile* to *order\_profile*. The order profile determines the validation process for the form. We'll go into this in the next section.

Lastly we set *mv\_cyber\_mode* to be *minivend\_test*. The *mv\_cyber\_mode* value determines which method will be used to charge a credit card. The value of *minivend\_test* tells it to use the internal test method which calculates a simple checksum against the card to determine whether 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 info. We are using all of the expected field names in this form so that no translation has to take place.

Have a look at the checkout page in your browser. The "Finalize!" link won't work yet, of course.

## 12.2. etc/profiles.order

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

Now we need to set up verification for the order form. This involves defining an order profile for the form. Create `etc/profiles.order` as follows and then we'll discuss the various features.

```
__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.) We then list the fields from the form which 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 useful if you have a multi–page ordering process, each of which you wish to validate as you go through them. The `__END__` pragma signals the end of this profile, after which you are free to begin another one.

In order to make your order profile effective, add the following *OrderProfile* directive to the bottom of `catalog.cfg`:

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

## 12.3. 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`. A simple example of this page is:

```
[include top]
[include left]
<p>You left something important out:</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[/page]
and fill the form out properly.</p>
```

```
[include bottom]
```

The important tag here is `[error]`. The output for this tag with these parameters is terse. The *all* parameter tells it to iterate through all of the errors reported from the failed verification. The *show\_var* parameter indicates that the failed variable should be displayed. For example, if the first name was left empty, *fname* would be shown. Then *show\_error* displays the actual error for the variable. The *joiner* parameter says that an HTML `<br>` tag should go between each error message, that is, each error should be on its own line. In more complex configurations, the `[error]` tag can be much more expressive.

## 12.4. Credit card processing

For this tutorial, we are going to implement a very simple order process. To this end, we will add one more directive 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 then erases the unencrypted copy from memory. We are using the standard option not to encrypt the number (in fact we aren't even going to define an encryption program) but to run the checksum verification on the number to verify that it is at least a potentially correct number. We will not be checking with a real payment processor to see if it is actually 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 wish to keep the number in memory so that it is available later in the process, when it is mailed as part of the order.

If this passes and all of the required fields are present, then the customer will be sent to the final page indicating success. Behind the scenes, Interchange will move on to send mail to the store owner (you).

## 12.5. etc/report

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

For it to be useful, the report will need to include the customer's name and address, as well as the items they ordered. The following is a simple report template which will report this information. Save it as `etc/report`.

```
Name: [value fname] [value lname]
Address: [value address1][if value address2]
```

## Interchange: Catalog–Building Tutorial

```
                [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 white space is relevant, unlike HTML. It is fairly straightforward, except that we have added the `[if]` tag to only include the optional second address line if the shopper filled it in.

One of the special properties of the `mv_credit_card_number` field is that Interchange specifically precludes it from getting saved so that it persists throughout the session. Thus it is unavailable to us in the `[value]` tag. The `[cgi]` tag can be used to circumvent this important security measure and get the value submitted from the last form.

**WARNING!** Obviously it would be a bad idea to send a real credit card number over an insecure channel like email. We are doing this purely for academic purposes. In a real configuration, you would absolutely want to encrypt the number in a secure fashion before emailing or storing it anywhere. You ignore this important security warning at your own peril.

### 12.6. special\_pages/receipt.html

Once the report has been run, Interchange will finish things on the customer's end by displaying a success screen. We are just going to show a nice "Thank you for ordering stuff from us" type of page. The default location for this page is `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[/page]</p>
[include bottom]
```

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

At this point you have a more–or–less fully dysfunctional store. Congratulations.



# 13. Enhancing the catalog

Now that we have a working catalog, we can go back and add little improvements here and there and test them incrementally. We'll walk through several together and then suggest more enhancements you can figure out on your own.

## 13.1. Price pictures

It's probably really been annoying you that the product prices aren't formatted as prices usually are. You may have even added a dollar sign (\$) before the `[loop-field price]` tag in your `pages/index.html` file to help out. But that still didn't make the thirty-seven dollar price of the "Ubiquitous diff eq final" test show up as "\$37.00" — instead it was simply "\$37". (Assuming you didn't change the price from our sample data and your system locale is the same as ours.) The way to deal with this is an Interchange feature called *price pictures*.

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

```
Locale en_US currency_symbol $
```

Restart Interchange and go look at your catalog. You'll notice little changed on the welcome page or the flypages, but in the shopping cart, all your prices should be formatted as U.S. dollars usually are: "1347.3" becomes "\$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 wherever prices show up. Make this 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 '-' or anything after it. The next line that starts with '+' replaces it.

---

A similar change to the `[item-field price]` tag in `pages/flypage.html` will fix that page. Now take a look in your browser. All your prices should look pretty.

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
```

That will make the currency symbol precede the currency value. A '0' setting will make the symbol come after the currency value.

```
Locale en_US mon_thousands_sep ,
```

That sets your thousands separator to a comma. You can set it to whatever you want.

```
Locale en_US mon_decimal_point .
```

That should be pretty obvious too. Many countries use a comma instead of a period to separate the integer from the decimal part.

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

## 13.2. Catalog variables

Interchange has provides a very useful feature that we haven't used yet: catalog variables. This is a way for us to set a variable to a certain value in our `catalog.cfg` file, then use it anywhere in our catalog pages. The *Variable* directive causes an Interchange catalog variable to be created, with the name taken 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, use a token with the pattern `__SOMENAME__`. Note that there are two underscore characters before the variable name and two more after the name, and that in place of the word `SOMENAME` you put the actual name of the variable. Interchange will replace the token with the variable's value before anything else is done to the page. The value can even include Interchange tags to be parsed.

## 13.3. A slightly more interesting page footer

Let's put a contact email address at the bottom of each page in case our loyal customers would like to get in touch with us. You could just add it to the footer, but by putting it in a variable, you can use it in contact pages as well. Then later on it's easy to change in one place and be reflected in many places. Set a catalog variable in `catalog.cfg` like this:

```
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>
```

Make sure to restart Interchange before reloading the page in your browser, since we've changed `catalog.cfg`.

Let's add another variable to our catalog. This time it will contain an Interchange tag that returns the current date in a nice format. Add this to `catalog.cfg`:

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

The `[tag]` tag is documented in the Interchange Tag Reference. The format strings inside `[tag time]` are documented in your system `strftime(3)` manual page. We won't discuss them further here.

Now add the variable to the boring `left` template piece:

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

Restart Interchange and try it out.

### 13.4. Fancy credit card expiration date selection

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

The following change to your `pages/checkout.html` gives you just such an enhancement. (You may want to read the explanation below before typing it in so 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 we generate a list of the months to choose from. We accomplish this by using a **[loop]** tag. In this case we are looping over an explicit list. We provide the list in the *list* parameter. Be careful with this, as it is sensitive to formatting—which the formatting of this document may not reflect. You want to make sure that the numbers are the first characters on each new line and that the elements are tab-separated. 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 element you want. Notice that the elements are zero-indexed. Each time through this loop Interchange will generate 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 we use embedded Perl 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 each of seven years in addition to the current year. It is not necessary at this point for you to understand this Perl code.

## 13.5. Sorting the product list

You may have noticed that the products listed on your welcome page are shown in the same order you entered them into `products/products.txt`. As you add more products, you'll want this list to show up in a nice, predictable order. We need to change the search parameters in `index.html`, which originally were:

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

You'll recall that 'ra' stands for 'return all' and 'fi' stands for file. Let's use the search parameter 'tf', which specifies the **sort field**. We 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`). Let's change the search in `index.html` to:

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

Now take a look. That's not quite what we wanted. The default ordering is done on a character–by–character basis, but we actually want to do a numeric sort here. We need to set 'to', the **sort order**, to 'n', for *numeric*:

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

That should work better. 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 it worked equally well to specify the sort field by number instead of name. We can also do a reverse alphabetical sort by description, silly as that may be:

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

Let's try narrowing our search down a bit. We'll no longer be returning all; instead, we'll give 'se', the **search spec**, and turn on 'su', which allows **substring matches**. Let's look only for products that have the word "test" in one of their fields somewhere, and sort the results by description, like this:

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

Which seems like something that would be better done in a search box for our store visitors. Before we go on, let's change this search back to the simple list, sorted by description:

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

### 13.6. A search box

Our customers would appreciate the ability to search for a test by SKU or part of the test description. Let's help them out. First, we'll add a search box to the hitherto unused left portion of our page layout. Make this 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 our search and pass the results to a still–nonexistent page called `pages/results.html`.

The `[set testname] ... [/set]` tags set an Interchange 'value' variable, which in this case we will use 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'.

Our search box will now appear on all our catalog pages, but we still need to create the search results page. Interchange will go to `pages/results.html` to display search results. Let's create it now:

```
[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>
    </table>
  [/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]Go to welcome page[/page]</p>
<p align=center>[page order]View shopping cart[/page]</p>
[include bottom]
```

The search results will all be contained within `[search–region] [/search–region]` tags. The text in the `[on–match] [/on–match]` container will only be displayed if there matches were found for the search, while text in the `[no–match] [/no–match]` will only be displayed 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.

That's all there is to it.

## 13.7. Default catalog page

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

```
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? Try:

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

And you get a nasty server error message. We can deal with this by adding a directive to `catalog.cfg`:

```
SpecialPage catalog index
```

Now restart Interchange and try the above URL again.

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

### 13.8. High–traffic changes

All through the tutorial we've created catalog pages that use the `[include]` tag to include template pieces into our pages. This has worked well for us, but there are a few drawbacks. One is that if we want to rename any of the template piece files, or move them out of the main catalog directory and into their own subdirectory, we would have to update the `[include]` tags on every page. To avoid this, we can set up 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 just change every instance of `[include top]` to `__TOP__`, and so on for each `[include]` tag. You may not feel like doing a bunch of search–and–replace on all the `.html` files you just created, but it's a good idea to use this capability in the next catalog you work on.

If you made all 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 all your pages up to date:

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

And so on, depending on your naming scheme.

Another concern is that every time a catalog page is viewed, each file in an `[include]` tag must be loaded from disk anew. In a test situation, this takes no noticeable amount of time at all. But on a very busy Interchange server, this can start to slow things down somewhat.

We can switch to a high–traffic mode that doesn't require that each template piece be read from disk every time the page is hit. Instead, all the pieces are read into variables once when Interchange is started and they remain in memory from then on. On very busy Interchange catalogs, this can speed things up noticeably. The only drawback is that you must restart the Interchange daemon whenever 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
```



## 14. Ideas for further enhancements

You can expand your skill with Interchange by adding more functionality to your test catalog. Here are some relatively 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 we 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.

---

Copyright 2001 Akopia, Inc. Freely redistributable under terms of the GNU General Public License.

