Catalog−Building Tutorial

Table of Contents

1. Purpose **1**

2. Before you begin **3**
2.1. Install Interchange and the demo catalog 3
2.2. The Interchange operating system user 3
2.3. Important directories 3
2.4. Your catalog URL 4
2.5. Starting or restarting Interchange 4
2.6. Tutorial assumptions 4

3. Building Your Catalog **7**
3.1. Create the link program 7
3.2. Create the tutorial catalog directory 7
3.3. Become the Interchange user 7
3.4. Go to the tutorial catalog directory 7
3.5. Create the session directory 8

4. Configuration files **9**
4.1. interchange.cfg 9
4.2. catalog.cfg 9

5. The products database table **11**
5.1. products/products.txt 11

6. Page templates **13**
6.1. top 13
6.2. left 13
6.3. bottom 13
6.4. The Interchange Tag Language 14

7. Creating a welcome page **15**
7.1. pages/index.html 15

8. Troubleshooting **17**

9. Displaying products **19**
9.1. Listing all products 19
9.2. pages/flypage.html 20
9.3. special_pages/missing.html 20

10. The shopping basket **23**
10.1. A link for ordering 23
10.2. pages/ord/basket.html 23

11. Order checkout **25**
11.1. pages/checkout.html 25
11.2. etc/profiles.order 27
11.3. special_pages/needfield.html 27

Table of Contents

11. Order checkout
11.4. Credit card processing....28
11.5. etc/report....28
11.6. special_pages/receipt.html....29

12. Enhancing the catalog....31
12.1. Price pictures....31
12.2. Catalog variables....32
12.3. A more interesting page footer....32
12.4. Advanced credit card expiration date selection....33
12.5. Sorting the product list....34
12.6. Adding a search box....35
12.7. The default catalog page....37
12.8. High−traffic changes....37
12.9. High traffic mode....38

13. Ideas for further enhancements....39

A. Catalog directory structure....41

B. Document history....43

1. Purpose

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

The simple catalog you create using this tutorial should give you a feel for the basic Interchange system. It should also be considered a stepping stone to a more complete and functional e−commerce system built with Interchange. The tutorial relies as much as possible on default settings to accentuate how Interchange works. It will use as few of Interchange's capabilities as possible, while still building a usable store. The resulting site will be simple but usable. The value of this tutorial is in the instruction that occurs along the way.

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

2. Before you begin

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

2.1. Install Interchange and the demo catalog

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

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

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

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

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

2.2. The Interchange operating system user

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

2.3. Important directories

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

- Interchange software directory
    - **RPM install:** /usr/lib/interchange
    - **Manual install as root:** /usr/local/interchange
    - **Manual install as regular user:** /home/username/interchange
- Catalogs directory
    - **RPM install:** /var/lib/interchange
    - **Manual install as root:** /usr/local/interchange/catalogs

- **Manual install as regular user:** /home/username/catalogs
- cgi−bin directory
  - **RPM install or source install as root:** /var/www/cgi−bin
  - **Manual install as root (locally installed web server):** /usr/local/htdocs, /opt/www, ...
  - **Manual install as regular user:** /home/username/public_html (with .cgi extension)

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

2.4. Your catalog URL

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

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

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

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

2.5. Starting or restarting Interchange

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

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

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

2.6. Tutorial assumptions

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

- **Interchange software directory:** /usr/lib/interchange
- **Catalogs directory:** /var/lib/interchange
- **cgi−bin directory:** /var/www/cgi−bin

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

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

3. Building Your Catalog

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

3.1. Create the link program

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

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

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

```
cp −p foundation tutorial
```

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

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

3.2. Create the tutorial catalog directory

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

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

3.3. Become the Interchange user

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

3.4. Go to the tutorial catalog directory

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

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

3.5. Create the session directory

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

4. Configuration files

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

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

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

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

4.1. interchange.cfg

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

```
Catalog   name   catalog_base_directory   link_url_path
```

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

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

Save the file.

4.2. catalog.cfg

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

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

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

```
Database  name  filename  format
```

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

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

This tells Interchange that you have a database table named 'products' that is described in a tab−delimited file named `products.txt`. You can describe an unlimited number of arbitrary database tables for the system to use this way. Interchange keeps a list of default tables called "Product Files," reflecting its e−commerce roots. You can specify all of the database tables that contain products by using the ***ProductFiles*** directive. There is no default for this, so you will have to specify your products table's name by adding the following line to `catalog.cfg`:

```
ProductFiles  products
```

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

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

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

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

5. The products database table

5.1. products/products.txt

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

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

The product database is handled as a special case since Interchange expects at least the description, price, and product ID (sku) fields. In other words, the `products.txt` file must at least contain fields named sku, price, and description. You can have other fields too, if you wish.

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

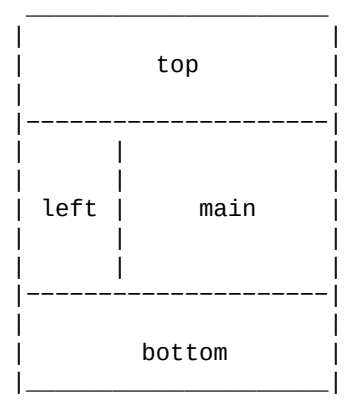
```
sku   description     price
4595  Nice Bio Test   275.45
2623  Stack of Econ Quizzes   1.24
0198  Really Hard Physics Test        1589.34
1299  Ubiquitous diff eq final        37.00
```

Note: When using tab−delimited files as we are, make sure you have exactly one tab between each field. Some text editors will use spaces to simulate tabs. Interchange expects actual ASCII tab characters; extra spaces or other characters will corrupt your data.

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

6. Page templates

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



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

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

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>
```

6.4. The Interchange Tag Language

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

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

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

7. Creating a welcome page

7.1. pages/index.html

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

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

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

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

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

---

8. Troubleshooting

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

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

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

9. Displaying products

9.1. Listing all products

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

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

. . .

</table>
```

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

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

. . .

[/loop]
```

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

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

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

Your finished page should look like this:

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

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

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

9.2. pages/flypage.html

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

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

```
[include top]
[include left]

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

[include bottom]
```

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

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

becomes:

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

9.3. special_pages/missing.html

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

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

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

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

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

10. The shopping basket

10.1. A link for ordering

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

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

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

10.2. pages/ord/basket.html

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

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

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

```
  [include top]
  [include left]

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

  <table cellpadding=5>

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

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

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

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

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

  </table>

  <hr>

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

  [include bottom]
```

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

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

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

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

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

11. Order checkout

11.1. pages/checkout.html

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

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

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

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

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

<table cellpadding=3>

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

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

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

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

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

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

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

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

</table>

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

<table cellpadding=3>

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

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

</table>

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

</form>

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

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

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

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

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

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

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

11.2. etc/profiles.order

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

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

```
__NAME__ order_profile

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

&fatal=yes
&final=yes

__END__
```

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

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

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

Watch for white space in front of the __NAME__ pragma, it can cause your profile to be ignored. Rember to restart Interchange for any changes to take effect.

11.3. special_pages/needfield.html

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

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

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

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

  [include bottom]
```

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

11.4. Credit card processing

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

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

  __END__
```

This issues two instructions to the credit card system.

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

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

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

11.5. etc/report

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

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

```
Name: [value fname] [value lname]
```

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

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


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

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

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

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

11.6. special_pages/receipt.html

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

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

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

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

12. Enhancing the catalog

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

12.1. Price pictures

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

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

```
Locale en_US currency_symbol $
```

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

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

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

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

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

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

```
Locale en_US p_cs_precedes 1
```

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

```
Locale en_US mon_thousands_sep ,
```

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

```
Locale en_US mon_decimal_point .
```

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

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

12.2. Catalog variables

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

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

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

12.3. A more interesting page footer

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

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

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

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

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

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

`catalog.cfg`:

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

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

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

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

Restart Interchange and view the page.

12.4. Advanced credit card expiration date selection

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

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

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

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

  </table>
```

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

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

12.5. Sorting the product list

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

12.6. Adding a search box

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

`left`:

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

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

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

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

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

```
[/search−region]
<hr>
<p align=center>[page index]Return to welcome page</a></p>
<p align=center>[page order]View shopping cart</a></p>
[include bottom]
```

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

12.7. The default catalog page

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

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

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

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

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

```
SpecialPage catalog index
```

Restart Interchange and try the above URL again.

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

12.8. High−traffic changes

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

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

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

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

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

And so on, depending on your naming scheme.

12.9. High traffic mode

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

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

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

13. Ideas for further enhancements

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

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

A. Catalog directory structure

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

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

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

B. Document history

October 2000. Conceived and written by Sonny Cook.

December 2000. Edited and expanded by Jon Jensen.

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

12 January 2001. First public release.

12 April 2002. Remove mention of obsolete Red Hat Linux 6−specific RPMs.
