Interchange Databases

Table of Contents

Table of Contents

1. Databases and Interchange

Interchange is database−independent, perhaps more so than almost any other powerful content management system.

Interchange can use GDBM, DB_File, SQL, LDAP, or in−memory databases. In most cases, these different database formats should operate the same when called by Interchange's access methods.

Also, most all of Interchange's core functions do not use hard−coded field names; virtually every field can have a configurable name.

Interchange does not require an external SQL database. If you have a small data set and do not want to integrate your own tool set, you cound use Interchange's internal database. However, the order management functions of Interchange will be slower and not as robust without an SQL database. SQL is strongly recommended for at least the `state`, `country`, `orderline`, `transactions`, and `userdb` tables. Any other tables that will have programmatic updates, such as `inventory`, will be best placed in SQL.

If you plan on using Interchange Admin UI, you should make the move to SQL. It provides easy import routines for text files that should replace text−file uploads.

Keeping a database in an SQL manager makes it easier to integrate Interchange with other tools. Interchange can be used to maintain a spreadsheet containing product information through modifying the file `products.txt` as needed. References to SQL, DBI, and DBD can be ignored.

1.1. Text Source Files

Interchange reads delimited text files to obtain its initial data. However, the text files are not the database. They are the source information for the database tables.

By default, all database source files are located in the `products` subdirectory of the catalog directory. The main products database is in the `products/products.txt` file in the supplied demo catalog.

Note: If you are using one of the internal database methods, any changes made to the ASCII source file will be reflected in the database in the next user session. If the product database contains less than a thousand records, updates will be instantaneous. If the product database is larger, updates will take longer. Use the `NoImport` reference tag to stop auto updating.

In the following configuration directive:

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

the `products` table will obtain its source information from the file `products.txt`. What is done with it depends on the type of underlying database being used. The different types and their behavior are described below:

GDBM

The database source file is checked to see if it is newer than the actual database file, `products.gdbm`. If it is, the database table is re−imported from the file.

This behavior can be changed in a few ways. If files should not be imported unless the `.gdbm` file disappears, set the `NoImport` directive:

```
NoImport  products
```

If the database source file is only to be imported at catalog start−up time, use the IMPORT_ONCE modifier:

```
Database products IMPORT_ONCE 1
```

GDBM is the default database type if the GDBM_File Perl module is installed (as it is on LINUX).

DB_File

The database source file is checked to see if it is newer than the actual database file, `products.db`. If it is, the database table is re−imported from the file. You can change this behavior in the same way as GDBM_File, described above.
DB_File is the default database type if the GDBM_File Perl module is not installed. This is common on FreeBSD. To specify DB_File as your database type, set it in `catalog.cfg` with a Database directive:

```
Database  products  DB_FILE   1
```

DBI/SQL

> If a file named `products.sql` is in the same directory as `products.txt`, the database table will not be imported from the ASCII source. If there is no `products.sql`, the following will occur: DBI/SQL imports will only happen at catalog configuration time.

Interchange will connect to the SQL database using the specified DSN. (DBI parameter meaning "Database Source Name".)
The table will be dropped with "DROP TABLE products;". This will occur without warning. NOTE: This can be prevented in several ways. See `NoImport External` or the SQL documentation for more information.
The table will be created. If there are COLUMN_DEF specifications in `catalog.cfg`, they will be used. Otherwise, the key (first field in the text file by default) will be created with a `char(16)` type and all other fields will be created as `char(128)`. The table creation statement will be written to the `error.log` file.
The text source file will be imported into the SQL database. Interchange will place the data in the columns. Data typing must be user−configured. This means that if "none" is placed in a field, and it is defined as a numeric type, the database import will not succeed. And if it does not succeed, the catalog will not become active.

In−Memory

Every time the catalog is configured, the `products.txt` file is imported into memory and forms the database. Otherwise, the database is not changed. The in−memory database is the default database if there is no GDBM_File or DB_File Perl module installed; specify it with:

```
Database   products   MEMORY   1
```

1.2. Interchange Database Conventions

This section describes naming and file usage conventions used with Interchange.

Note: Throughout the documentation, the following terms and their definitions are used interchangeably:

---

key, code

A reference to the database key. In Interchange, this is usually the product code or SKU, which is the part number for the product. Other key values may be used to generate relationships to other database tables. It is recommended that the key be the first column of the ASCII source file, since Interchange's import, export, and search facilities rely on this practice.

field, column

The vertical row of a database. One of the columns is always the key and it is usually the first one.

table, database

A table in the database. Because Interchange has evolved from a single−table database to an access method for an unlimited number of tables (and databases, for that matter), a table will occasionally be referred to as a database. The only time the term database refers to something different is when describing the concept as it relates to SQL, where a database contains a series of tables. While Interchange cannot create SQL databases, it can drop and create tables with that database if given the proper permissions.

If necessary, Interchange can read the data to be placed in tables from a standard ASCII−delimited file. All of the ASCII source files are kept in the products directory, which is normally in the catalog directory (where catalog.cfg is located). The ASCII files can have ^M (carriage return) characters, but must have a new line character at the end of the line to work. NOTE: Mac users uploading files must use ASCII mode, not binary mode.

Interchange's default ASCII delimiter is TAB.

Note: The items must be separated by a single delimiter. The items in this document are lined up for reading convenience.

TAB

Fields are separated by `^I` characters. No whitespace is allowable at the beginning of the line.

```
code    description             price   image
SH543   Men's fine cotton shirt 14.95   shirts.jpg
```

PIPE

Fields are separated by pipe `|` characters. No whitespace is allowable at the beginning of the line.

```
code|description|price|image
SH543|Men's fine cotton shirt|14.95|shirts.jpg
```

CSV

Fields are enclosed in quotes, separated by commas. No whitespace should be at the beginning of the line.

```
"code","description","price","image"
```

```
"SH543","Men's fine cotton shirt","14.95","shirts.jpg"
```

Note: Using the default TAB delimiter is recommended if you plan on searching the ASCII source file of the database. PIPE works fairly well, but CSV delimiter schemes might cause problems with searching.

IMPORTANT NOTE: Field names are usually case−sensitive. Use consistency when naming or you might encounter problems. All lower or all upper case names are recommended.

Interchange uses one mandatory database, which is referred to as the products database. In the supplied demo catalog, it is called products and the ASCII source is kept in the file `products.txt` in the products directory. This is also the default file for searching with the THE SEARCH ENGINE. Interchange also has a two of standard, but optional, databases that are in fixed formats:
shipping.asc

The database of shipping options that is accessed if the `CustomShipping` directive is in use. This is a fixed−format database, and must be created as specified. For more information, see the Shipping ITL tag in the *Interchange Tag Reference Guide*.
salestax.asc

The database of sales tax information if the `[salestax]` tag is to be used. A default is supplied. NOTE: Caution, these things change! This is a fixed−format database, and must be created as specified. See Sales Tax.
These are never stored in SQL or DBM.

1.3. The Product Database

Each product being sold should be given a product code, usually referred to as SKU, a short code that identifies the product on the ordering page and in the catalog. The products.txt file is a ASCII−delimited list of all the product codes, along with an arbitrary number of fields which must contain at least the fields `description` and `price` (or however the `PriceField` and `DescriptionField` directives have been set). Any additional information needed in the catalog can be placed in any arbitrary field. See Interchange Database Capability for details on the format.

Field names can be case−sensitive depending on the underlying database type. Unless there are fields with the names "description" and "price" field, set the `PriceField` and `DescriptionField` directives to use the `[item−price]` and `[item−description]` tags.

The product code, or SKU, must be the first field in the line, and must be unique. Product codes can contain the characters **A−Za−z0−9**, along with hyphen (−), underscore (_), pound sign/hash mark (#), slash (/), and period (.). Note that slash (/) will interfere with on−the−fly page references. Avoid if at all possible.

The words should be separated by one of the approved delimiting schemes (TAB, PIPE, or CSV), and are case−sensitive in some cases. If the case of the "description" or "price" fields have been modified, the `PriceField` and `DescriptionField` directives must be appropriately set.

Note: CSV is not recommended as the scheme for the products database. It is much slower than TAB− or PIPE−delimited, and dramatically reduces search engine functionality. No field−specific searches are possible. Using CSV for any small database that will not be searched is fine.

IMPORTANT NOTE: The field names must be on the first line of the `products.txt` file. These field names must match exactly the field names of the `[item-field]` tags in the catalog pages, or the Interchange server will not access them properly. Field names can contain the characters A-Za-z0-9 and underscore (_).

More than one database may be used as a products database. If the catalog directive, ProductFiles, is set to a space-separated list of valid Interchange database identifiers, those databases will be searched (in the order specified) for any items that are ordered, or for product information (as in the `[price code]` and `[field code]` tags).

When the database table source file (i.e., products.txt) changes after import or edit, a DBM database is re-built upon the next user access. No restart of the server is necessary.

If changing the database on-the-fly, it is recommended that the file be locked while it is being modified. Interchange's supplied import routines do this.

1.4. Multiple Database Tables

Interchange can manage an unlimited number of arbitrary database tables. They use the TAB delimiter by default, but several flexible delimiter schemes are available. These are defined by default:

```
Type 1      DEFAULT - uses default TAB delimiter
Type 2      LINE
            Each field on its own line, a blank line
            separates the record. Watch those carriage
            returns! Also has a special format when CONTINUE
            is set to be NOTES.
Type 3      %%
            Fields separated by a \n%%\n combination, records by
            \n%%%\n (where \n is a newline). Watch those carriage
            returns!
Type 4      CSV
Type 5      PIPE
Type 6      TAB
Type 7      reserved
Type 8      SQL
Type 9      LDAP
```

The databases are specified in `Database` directives, as:

```
Database    arbitrary arbitrary.csv CSV
```

This specifies a Type 4 database, the ASCII version of which is located in the file `arbitrary.csv`, and the identifier it will be accessed under in Interchange is "arbitrary." The DBM file, if any, will be created in the same directory if the ASCII file is newer, or if the DBM file does not exist. The files will be created as `arbitrary.db` or `arbitrary.gdbm`, depending on DBM type.

The `identifier` is case sensitive, and can only contain characters in the class [A-Za-z0-9_]. Fields are accessed with the [item_data `identifier` field] or [data `identifier` field key] elements. NOTE: Use of lower-case letters is strongly recommended.

If one of the first six types is specified, the database will automatically be built in the default Interchange DB style. The type can be specified with DB_FILE, GDBM, or MEMORY, if the type varies from that default. They will coexist with an unlimited number of DBI databases of different types.

In addition to the database, the session files will be kept in the default format, and are affected by the following actions.

The order of preference is:

GDBM

This uses the Perl `GDBM_File` module to build a GDBM database. The following command will indicate if GDBM is in Perl:

```
perl −e 'require GDBM_File and print "I have GDBM.\n"'
```

Installing GDBM_File requires rebuilding Perl after obtaining the GNU GDBM package, and is beyond the scope of this document. LINUX will typically have this by default; most other operating systems will need to specifically build in this capability.

DB_File (Berkeley DB)

This uses the `DB_File` module to build a Berkeley DB (hash) database. The following command will indicate if DB_File is in Perl:

```
perl −e 'require DB_File and print "I have Berkeley DB.\n"'
```

Installing `DB_File` requires rebuilding Perl after obtaining the Berkeley DB package, and is beyond the scope of this document. BSDI, FreeBSD, and LINUX will typically have it by default; most other operating systems will need to specifically build this in.
If using `DB_File`, even though `GDBM_File` is in Perl, set the environment variable MINIVEND_DBFILE to a true (non−zero, non−blank) value:

```
# csh or tcsh
setenv MINIVEND_DBFILE 1

# sh, bash, or ksh
MINIVEND_DBFILE=1 ; export MINIVEND_DBFILE
```

Then, re−start the server.
Or, to set a particular table to use Berkeley DB, the DB_FILE class in `catalog.cfg` can be specified:

```
Database arbitrary  DB_FILE  1
```

In−memory

This uses Perl hashes to store the data directly in memory. Every time the Interchange server is restarted, it will re−import all in−memory databases for every catalog.
If this is used, despite the presence of `GDBM_File` or `DB_File`, set the environment variable MINIVEND_NODBM as above or specify the memory type in the Database directive:

```
Database arbitrary  MEMORY  1
```

Note: The use of memory databases is not recommended.

1.5. Character Usage Restrictions

To review, database identifiers, field names, and product codes (database keys) are restricted in the characters they may use. The following table shows the restrictions:

```
                                Legal characters
                                --------------------
 Database identifiers           A-Z a-z 0-9 _
 Field names                    A-Z a-z 0-9 _
 Database keys (product code/SKU) A-Z a-z 0-9 _ # - . /
 Database values                Any (subject to field/record delimiter)
```

Some SQL databases have reserved words which cannot be used as field names; Interchange databases do not have this restriction.

For easy HTML compatibility, it is not recommended that a / be used in a part number if using the flypage capability. It can still be called [page href=flypage arg="S/KU"].

1.6. Database Attributes

Especially in SQL databases, there are certain functions that can be set with additional database attributes. For text import, the CONTINUE extended database import attribute allows additional control over the format of imported text.

Note: CONTINUE applies to all types except CSV. (Do not use NOTES unless using type LINE.)

CONTINUE

One of UNIX, DITTO, LINE, NONE, or NOTES. The default, NONE, is to simply split the line/record according to the delimiter, with no possible spanning of records. Setting CONTINUE to UNIX appends the next line to the current when it encounters a backslash (\) at the end of a record, just like many UNIX commands and shells.
DITTO is invoked when the key field is blank. It adds the contents of following fields to the one above, separated by a new line character. This allows additional text to be added to a field beyond the 255 characters available with most spreadsheets and flat-file databases.
Example in catalog.cfg:

```
Database products products.txt  TAB
Database products CONTINUE      DITTO
```

Products.asc file:

```
code     price     description
00-0011  500000    The Mona Lisa, one of the worlds great masterpieces.
                   Now at a reduced price!
```

The description for product 00-0011 will contain the contents of the `description` field on both lines, separated by a new line.

Note: Fields are separated by tabs, formatted for reading convenience.

This will work for multiple fields in the same record. If the field contains any non−empty value, it will be appended.
LINE is a special setting so a multi−line field can be used. Normally, when using the LINE type, there is only data on one line separated by one blank line. When using CONTINUE LINE, there may be some number of fields which are each on a line, while the last one spans multiple lines up until the first blank line.
Example in catalog.cfg:

```
Database products products.txt  LINE
Database products CONTINUE      LINE
```

Products.asc file:

```
code
price
description

00−0011
500000
The Mona Lisa, one of the worlds great masterpieces.
Now at a reduced price!

00−0011a
1000
A special frame for the Mona Lisa.
```

NOTES reads a Lotus Notes "structured text" file. The format is any number of fields, all except one of which must have a field name followed by a colon and then the data. There is optional whitespace after the colon. Records are separated by a settable delimiting character which goes on a line by itself, much like a "here document." By default, it is a form feed (^L) character. The final field begins at the first blank line and continues to the end of the record. This final field is named `notes_field`, unless set as mentioned below. Interchange reads the field names from the first paragraph of the file. The key field should be first, followed by other fields in any order. If one (and only one) field name has whitespace, then its name is used for the `notes_field`. Any characters after a space or TAB are used as the record delimiter.
If there are none, then the delimiter returns to the default form feed (^L) and the field name reverts to `notes_field`. The field in question will be discarded, but a second field with whitespace will cause an import error. Following records are then read by name, and only fields with data in them need be set. Only the `notes_field` may contain a new line. It is always the last field in the record, and begins at the **first** blank line.
The following example sets the delimiter to a tilde `(~)` and renames the `notes_field` to `description`.
Example in catalog.cfg:

```
Database products products.txt  LINE
Database products CONTINUE      NOTES
```

Products.asc file:

```
code
title
price
image
```

```
description ~
size
color

title: Mona Lisa
price: 500000
code: 00-0011
image: 00-0011.jpg

The Mona Lisa, one of the worlds great masterpieces.
Now at a reduced price!
~
title: The Art Store T-Shirt
code: 99-102
size: Medium, Large*, XL=Extra Large
color: Green, Blue, Red, White*, Black
price: 2000

Extra large 1.00 extra.
~
```

EXCEL

Microsoft Excel is a widely-used tool to maintain Interchange databases, but has several problems with its standard TAB-delimited export, like enclosing fields containing commas in quotes, generating extra carriage returns embedded in records, and not including trailing blank fields. To avoid problems, use a text-qualifier of none.
Set the EXCEL attribute to 1 to fix these problems on import:

```
Database products EXCEL 1
```

This is normally used only with TAB-delimited files.

LARGE

Interchange databases containing many records can result in a noticeable slowdown when displayed by the UI. Set the LARGE attribute to 1 to avoid this problem:

```
Database transactions LARGE 1
```

In this case the UI supplies only input boxes to search records in the database instead of drawing all the records from the database, sorting them and creating more lists.

1.7. Dictionary Indexing With INDEX

Interchange will automatically build index files for a fast binary search of an individual field. This type of search is useful for looking up the author of a book based on the beginning of their last name, a book title based on its beginning, or other similar situations.

Such a search requires a dictionary ordered index with the field to be searched contained in the first field and the database key (product code) in the second field. If the `INDEX field` modifier is specified, Interchange will build the index upon database import:

```
Database  products  products.txt   TAB
Database  products  INDEX          title
```

If the `title` field is the fourth column in the `products` database table, a file `products.txt.4` will be built, containing two tab−separated fields something like:

```
American Gothic   19−202
Mona Lisa         00−0011
Sunflowers        00−342
The Starry Night  00−343
```

Options can be appended to the field name after a colon (:). The most useful will be `f`, which does a case−insensitive sort. The `mv_dict_fold` option must be added to the search in this case.

Another option is `c`, which stands for "comma index." To index on comma−separated sub−fields within a field, use the :c option:

```
Database  products  products.txt   TAB
Database  products  INDEX          category:c
```

This can get slow for larger databases and fields. Interchange will split the field on a comma (stripping surrounding whitespace) and make index entries for each one. This allows multiple categories in one field while retaining the fast category search mechanism. It might also be useful for a `keywords` field.

The fast binary search is described in greater detail in THE SEARCH ENGINE below.

1.8. MEMORY for Memory−Only Databases

Interchange's memory−based databases are the fastest possible way to organize and store frequently used data. To force a database to be built in memory instead of DBM, use the MEMORY modifier:

```
Database  country  country.asc   TAB
Database  country  MEMORY        1
```

Obviously, large tables will use a great deal of memory, and the data will need to be re−imported from the ASCII source file at every catalog reconfiguration or Interchange restart. The big advantage of using MEMORY is that the database remains open at all times and does not need to be reinitialized at every connect. Use it for smaller tables that will be frequently accessed.

Memory tables are read only −− the MEMORY modifier forces IMPORT_ONCE.

1.9. IMPORT_ONCE

The IMPORT_ONCE modifier tells Interchange not to re−import the database from the ASCII file every time it changes. Normally, Interchange does a comparison of the database file modification time with the ASCII source every time it is accessed, and if the ASCII source is newer it will re−import the file.

IMPORT_ONCE tells it only to import on a server restart or catalog reconfiguration:

```
Database  products  products.txt   TAB
Database  products  IMPORT_ONCE    1
```

SQL databases don't normally need this. They will only be imported once in normal operation. Also see `NoImport` for a way to guarantee that the table will never be imported.

IMPORT_ONCE is always in effect for MEMORY databases. A catalog reconfiguration is required to force a change.

1.10. MIRROR

Additionally, you can have two tables, the regular table and the memory table by adding to the definition files:

```
Database country_memory country_memory.txt TAB
Database country_memory MIRROR          country
Database country_memory MEMORY          1
```

1.11. SQL/DBI parameters

1.11.1. AUTO_SEQUENCE

Tells Interchange to use a SQL sequence to number new database items inserted into the database.

If you have Interchange create the table, then you need to do:

```
Database foo foo.txt dbi:mysql:test
Database foo AUTO_SEQUENCE foo_seq
```

Then on MySQL, Pg, or Oracle, Interchange will create an integer key type and a sequence (or AUTO_INCREMENT in MySQL) to maintain the count.

1.11.2. AUTO_SEQUENCE_MAXVAL

Sets the MAXVAL to have in an AUTO_SEQUENCE counter:

```
Database foo AUTO_SEQUENCE_MAXVAL  1000000
```

1.11.3. AUTO_SEQUENCE_MINVAL

Sets the MINVAL to have in an AUTO_SEQUENCE counter:

```
Database foo AUTO_SEQUENCE_MINVAL  10
```

1.11.4. AUTO_SEQUENCE_START

Sets the starting value for an AUTO_SEQUENCE counter:

```
Database foo AUTO_SEQUENCE_START  1000
```

1.11.5. COMPOSITE_KEY

If you are using a DBI table with composite keys, where two or more fields combine to make the unique identifier for a record, you must tell Interchange so it can request data in the right way. To do this, set:

```
Database  product_spec product_spec.asc dbi:mysql:foobase
Database  product_spec COMPOSITE_KEY sku feature
```

```
Database  product_spec COLUMN_DEF    "sku=varchar(32)"
Database  product_spec COLUMN_DEF    "feature=varchar(128)"
```

If you want to create a custom index for the table, do so. If you don't specify a POSTCREATE or INDEX parameter for the table, Interchange will create a unique index with all composite key elements at table creation time.

1.11.6. DSN

The data source name (DSN) for the database. It is beyond the scope of this document to describe this in detail.

Normally this is set as the type in the initial `Database` configuration line, i.e.

```
Database  foo  foo.txt  dbi:mysql:foobase
```

This has the same effect:

```
Database  foo  foo.txt  SQL
Database  foo  DSN      dbi:mysql:foobase
```

Some other examples of DSN specs:

```
Database  foo  DSN  dbi:mysql:host=db.you.com;database=foobase
Database  foo  DSN  dbi:Pg:dbname=foobase
Database  foo  DSN  dbi:Oracle:host=myhost.com;sid=ORCL
```

1.11.7. HAS_TRANSACTIONS

Informs Interchange that the SQL database in use has commit() and rollback() for transactions. For PostgreSQL and Oracle this should be set properly to 1 −− for MySQL and other databases you have to set it.

1.11.8. HAS_LIMIT

Informs Interchange that the SQL database in use has as the LIMIT extension to SQL to limit return from queries. Should be set properly by default for MySQL, PostgreSQL, and Oracle.

1.11.9. POSTCREATE

One or more SQL statements that should be performed after Interchange creates a table.

```
Database foo POSTCREATE "create unique index foo_idx on foo(key1,key2)"
Database foo POSTCREATE "create index mulkey_idx on foo(mulkey)"
```

1.11.10. PRECREATE

One or more SQL statements that should be performed before Interchange creates a table.

```
Database foo POSTCREATE "drop table foobackup"
Database foo POSTCREATE "alter table foo rename to foobackup"
```

1.11.11. REAL_NAME

Sometimes it may be convenient to have a table named a consistent value in Interchange despite its name in the underlying database. For instance, two divisions of a company may share orders but have different `products` tables. You can tell Interchange to name the table `products` for its purposes, but use the `products_a` table for SQL statements:

```
Database products REAL_NAME products_a
```

Of course if you have SQL queries that are passed verbatim to Interchange (i.e. the [query ...] tag) you must use the REAL_NAME in those.

1.12. Importing in a Page

To add a data record to a database as a result of an order or other operation, use Interchange's `[import ...]` tag.

[import table type*] RECORD [/import]

Named parameters:

```
[import table=table_name
        file=filename*
        type=(TAB|PIPE|CSV|%%|LINE)*
        continue=(NOTES|UNIX|DITTO)*
        separator=c*]
```

Import one or more records into a database. The `type` is any of the valid Interchange delimiter types, with the default being TAB. The table must already be a defined Interchange database table. It cannot be created on−the−fly. If on−the−fly functionality is need, it is time to use SQL.

The import type selected need not match the type the database was specified. Different delimiters may be used.

The `type` of `LINE` and `continue` setting of `NOTES` is particularly useful, for it allows fields to be named and not have to be in any particular order of appearance in the database. The following two imports are identical in effect:

```
[import table=orders]
         code: [value mv_order_number]
shipping_mode: [shipping−description]
       status: pending
[/import]

[import table=orders]
shipping_mode: [shipping−description]
status:        pending
code:          [value mv_order_number]
[/import]
```

The `code` or key must always be present, and is always named `code`. If `NOTES` mode is not used, the fields must be imported in the same order as they appear in the ASCII source file.

The `file` option overrides the container text and imports directly from a named file based in the catalog directory. To import from `products.txt`, specify `file="products/products.txt"`. If the NoAbsolute directive is set to `Yes` in interchange.cfg, only relative path names will be allowed.

The `[import ....] TEXT [/import]` region may contain multiple records. If using `NOTES` mode, a separator must be used, which, by default, is a form−feed character (^L). See Import Attributes for more information.

1.13. Exporting from a Database

To export an existing database to a file to its text file, suitable for full−text search by Interchange, use Interchange's UI create a page that contains a `[export table=TABLENAME]` ITL tag (ExportTag).

1.14. Write Control

Interchange databases can be written in the normal course of events, either using the `[import ...]` tag or with a tag like `[data table=table column=field key=code value=new−value]`. To control writing of a global database, or to a certain catalog within a series of subcatalogs, or make one read only, see the following:

To enable write control:

```
Database   products  WRITE_CONTROL  1
```

Once this is done, to make a database read only, which won't allow writing even if `[tag flag write]products[/tag]` is specified:

```
Database   products  READ_ONLY  1
```

To have control with `[tag flag write]products[/tag]`:

```
Database   products  WRITE_TAGGED  1
```

To limit write to certain catalogs, set:

```
Database   products  WRITE_CATALOG  simple=0, sample=1
```

The "simple" catalog will not be able to write, while "sample" will if `[tag flag write]products[/tag]` is enabled. If a database is to always be writable, without having to specify `[tag flag write] ... [/tag]`, then define:

```
Database   products  WRITE_ALWAYS  1
```

The default behavior of SQL databases is equivalent to WRITE_ALWAYS, while the default for GDBM_File, DB_File, and Memory databases is equivalent to:

```
Database   products  WRITE_CONTROL 1
Database   products  WRITE_TAGGED  1
```

1.15. Global Databases

If a database is to be available to all catalogs on the Interchange server, it may be defined in `interchange.cfg`. Any catalog running under that server will be able to use it. It is writable by any catalog unless WRITE_CONTROL is used.

2. SQL Support

Interchange can use any of a number of SQL databases through the powerful Perl DBI/DBD access methods. This allows transparent access to any database engine that is supported by a DBD module. The current list includes mSQL, MySQL, Solid, PostgreSQL, Oracle, Sybase, Informix, Ingres, Dbase, DB2, Fulcrum, and others. Any ODBC (with appropriate driver) should also be supported.

No SQL database is included with Interchange, but there are a number widely available on the Internet. Most commonly used with Interchange are PostgreSQL, MySQL, and Oracle. It is beyond the scope of this document to describe SQL or DBI/DBD. Sufficient familiarity is assumed.

In most cases, Interchange cannot perform administrative functions, like creating a database or setting access permissions. This must be done with the tools provided with a SQL distribution. But, if given a blank database and the permission to read and write it, Interchange can import ASCII files and bootstrap from there.

2.1. SQL Support via DBI

The configuration of the DBI database is accomplished by setting attributes in additional Database directives after the initial defining line as described above. For example, the following defines the database **arbitrary** as a DBI database, sets the data source (DSN) to an appropriate value for an mSQL database named `minivend` on port 1114 of the local machine:

```
Database arbitrary arbitrary.asc SQL
Database arbitrary DSN           dbi:mSQL:minivend:localhost:1114
```

As a shorthand method, include the DSN as the type:

```
Database arbitrary arbitrary.asc dbi:mSQL:minivend:localhost:1114
```

Supported configuration attributes include (but are not limited to):

DSN

A specification of the DBI driver and its data source. To use the DBD::mSQL driver for DBI, use:

```
dbi:mSQL:minivend:othermachine.my.com:1112
```

where mSQL selects the driver (case IS important), `minivend` selects the database, `othermachine.my.com` selects the host, and 1112 is the port. On many systems, `dbi:mSQL:minivend` will work fine. Of course, the `minivend` database must already exist.
This is the same as the DBI_DSN environment variable, if the DSN parameter is not set. Then, the value of DBI_DSN will be used to try and find the proper database to connect to.

USER

The user name used to log into the database. It is the same as the environment variable **DBI_USER**. If a user name is not needed, just don't set the USER directive.

PASS

The password used to log into the database. It is the same as the environment variable **DBI_PASS**. If a password is not needed, just don't set the PASS directive.

COLUMN_DEF

A comma−separated set of lines in the form NAME=TYPE(N), where NAME is the name of the field/column, TYPE is the SQL data type reference, and N is the length (if needed). Most Interchange fields should be the fixed−length character type, something like char(128). In fact, this is the default if a type is not chosen for a column. There can be as many lines as needed. This is not a DBI parameter, it is specific to Interchange.

NAME

A space−separated field of column names for a table. Normally not used. Interchange should resolve the column names properly upon query. Set this if a catalog errors out with "dbi: can't find field names" or the like. The first field should always be **code**. This is not a DBI parameter, it is specific to Interchange. All columns must be listed, in order of their position in the table.

NUMERIC

Tells Interchange not to quote values for this field. It allows numeric data types for SQL databases. It is placed as a comma−separated field of column names for a table, in no particular order. This should be defined if a numeric value is used because many DBD drivers do not yet support type queries.

UPPERCASE

Tells Interchange to force field names to UPPER case for row accesses using the `[item−data ...]`, `[loop−data ...]`, `[item−field ...`, etc. Typically used for Oracle and some other SQL implementations.

DELIMITER

A Interchange delimiter type, either TAB,CSV,PIPE,%%,LINE or the corresponding numeric type. The default for SQL databases is TAB. Use DELIMITER if another type will be used to import. This is not a DBI parameter. It is specific to Interchange.

KEY

The keying default of `code` in the first column of the database can be changed with the KEY directive. Don't use this unless prepared to alter all searches, imports, and exports accordingly. It is best to just accept the default and make the first column the key for any Interchange database.

ChopBlanks, LongReadLen, LongTruncOK, RaiseError, etc.

Sets the corresponding DBI attribute. Of particular interest is `ChopBlanks`, which should be set on drivers which by default return space−padded fixed−length character fields (Solid is an example).
The supported list as of this release of Interchange is:

```
ChopBlanks
CompatMode
LongReadLen
```

```
        LongTruncOk
        PrintError
        RaiseError
        Warn
```

Issue the shell command `perldoc DBI` for more information.

Here is an example of a completely set up DBI database on MySQL, using a comma−separated value input, setting the DBI attribute `LongReadLen` to retrieve an entire field, and changing some field definitions from the default char(128):

```
Database   products  products.csv  dbi:mysql:minivend
Database   products  USER          minivend
Database   products  PASS          nevairbe
Database   products  DELIMITER     CSV

# Set a DBI attribute
Database   products  LongReadLen   128

# change some fields from the default field type of char(128)
# Only applies if Interchange is importing from ASCII file
# If you set a field to a numeric type, you must set the
# NUMERIC attribute
Database   products  COLUMN_DEF    "code=char(20) NOT NULL primary key"
Database   products  COLUMN_DEF    price=float, discount=float
Database   products  COLUMN_DEF    author=char(40), title=char(64)
Database   products  COLUMN_DEF    nontaxable=char(3)
Database   products  NUMERIC       price
Database   products  NUMERIC       discount
```

MySQL, DBI, and DBD::mysql must be completely installed and tested, and have created the database `minivend`, for this to work. Permissions are difficult on MySQL. if having trouble, try starting the MySQL daemon with `safe_mysqld −−skip−grant−tables &` for testing purposes.

To change to ODBC, the only changes required might be:

```
Database products  DSN         dbi:ODBC:TCP/IP localhost 1313
Database products  ChopBlanks  1
```

The DSN setting is specific to a ODBC setup. The `ChopBlanks` setting takes care of the space−padding in Solid and some other databases. It is not specific to ODBC. Once again, DBI, DBD::ODBC, and the appropriate ODBC driver must be installed and tested.

2.2. SQL Access Methods

An Interchange SQL database can be accessed with the same tags as any of the other databases can. Arbitrary SQL queries can be passed with the `[query sql="SQL STATEMENT"]` ITL tag.

```
[query
    ml=10
    more=1
    type=list
    sp="@@MV_PAGE@@"
    sql=|
        SELECT  sku, description
        FROM    products
```

```
            WHERE   somecol
                                        BETWEEN '[filter sql][cgi from][/filter]'
                                        AND '[filter sql][cgi to][/filter]'
            AND     someothercol = '[filter sql][cgi whatever][/filter]'
            ORDER BY sku
        |]
        [list]
            sku=[sql-code] - desc=[sql-param description]<br>
        [/list]
        [on-match]
            Something was found<br>
        [/on-match]
        [no-match]
            Nothing was found<br>
        [/no-match]
        [more-list]
                <br>[matches]<br>
        [/more-list]
    [/query]
```

Not the filter for [cgi foo] values, which prevent single quotes (') from destroying the query.

2.3. Importing from an ASCII File

When importing a file for SQL, Interchange by default uses the first column of the ASCII file as the primary key, with a `char(16)` type, and assigns all other columns a `char (128)` definition. These definitions can be changed by placing the proper definitions in COLUMN_DEF Database directive attribute:

```
Database  products  COLUMN_DEF  price=char(20), nontaxable=char(3)
```

This can be set as many times as desired, if it will not fit on the line.

```
Database  products  COLUMN_DEF  price=char(20), nontaxable=char(3)
Database  products  COLUMN_DEF  description=char(254)
```

To create an index automatically, append the information when the value is in quotes:

```
Database  products  COLUMN_DEF  "code=char(14) primary key"
```

The field delimiter to use is TAB by default, but can be changed with the Database DELIMITER directive:

```
Database  products products.csv dbi:mSQL:minivend:localhost:1114
Database  products DELIMITER  CSV
```

To create other secondary keys to speed sorts and searches, do so in the COLUMN_DEF:

```
Database  products COLUMN_DEF  "author=char(64) secondary key"
```

Or use external database tools. NOTE: Not all SQL databases use the same index commands.

To use an existing SQL database instead of importing, set the NoImport directive in catalog.cfg to include any database identifiers not to be imported:

```
  NoImport  products inventory
```

WARNING: If Interchange has write permission on the products database, be careful to set the NoImport directive or create the proper .sql file. If that is not done, and the database source file is changed, the SQL database could be overwritten. In any case, always back up the database before enabling it for use by Interchange.

---

3. Managing DBM Databases

3.1. Making the Database

The DBM databases can be built offline with the `offline` command. The directory to be used for output is specified either on the command line with the `-d` option, or is taken from the `catalog.cfg` directive `OfflineDir` -- `offline` in the catalog directory by default. The directory must exist. The source ASCII files should be present in that directory, and the DBM files are created there. Existing files will be overwritten.

```
offline -c catalog [-d offline_dir]
```

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

3.2. Updating Individual Records

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

The database is specified with the `-n` option, or is 'products' by default.

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

```
update -c catalog -f price 25.00
```

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

```
update -c catalog -f price -f comment 25.00 "That pitchfork couple"
```

The following takes input from `file`, which must be formatted exactly like the original database, and adds/corrects any records contained therein.

```
update -c catalog -i file
```

Invoke the command without any arguments for a usage message describing the options.

4. Other Database Capabilities

Interchange has a number of other options that can affect operation of or operations on a defined database.

4.1. Search Modification

Normally, Interchange can search any database and will return all records that match the search specification. Some attributes affect this.

4.1.1. HIDE_FIELD

When set to a field name, i.e.:

```
Database  sometable  HIDE_FIELD  inactive
```

Interchange will not return records that have that field (in the example, c<inactive>) set to a true (non−blank, non−zero) value.

4.1.2. NO_SEARCH

An indication that the database should not be searchable by default. Used to determine the default search files for a product searc.

```
Database  sometable  NO_SEARCH  1
```

In the `foundation` demo, this is used to prevent the `options` table from being searched for products.

4.2. Indexing

You can indicate that a database should be indexed on a field with the INDEX modifier:

```
Database sometable  INDEX  category
```

This will create an ASCII index on every import, and will also create an index on the field at SQL creation time.

If you wish to create SQL indices at table creation time *without* creating an ASCII index, use the NO_ASCII_INDEX parameter:

```
Database sometable  NO_ASCII_INDEX  1
```

Of course you can create a SQL index manually at any time via your SQL toolset.

5. The Search Engine

Interchange implements a search engine which will search the product database (or any other file) for items based on customer input. It uses either forms or link−based searches that are called with the special page name `scan`. The search engine uses many special Interchange tags and variables.

If the search is implemented in a link or a form, it will always display formatted results on the results page, an Interchange page that uses some combination of the `[search−region]`, `[search−list]`, `[more−list]`, `[more]`, and other Interchange tags to format and display the results. The search results are usually a series of product codes/SKUs or other database keys, which are then iterated over similar to the `[item−list]`.

Note: Examples of search forms and result pages are included in the demos.

Two search engine interfaces are provided, and five types of searching are available. The default is a text−based search of the first products database source file (i.e., products.txt). A binary search of a dictionary−ordered file can be specified. An optional Glimpse search is enabled by placing the command specification for Glimpse in the `catalog.cfg` directive `Glimpse`. There is a range−based search, used in combination with one of the above. And finally, there is a fully−coordinated search with grouping.

The default, a text based search, sequentially scans the lines in the target file. By default it returns the first field (delineated by the delimiter for that database) for every line matching the search specification. This corresponds to the product code, which is then used to key specific accesses to the database.

The text−based search is capable of sophisticated field−specific searches with fully−independent case−sensitivity, substring, and negated matching.

5.1. The Search Form

A number of variables can be set on search forms to determine which search will be used, what fields in the database it will search, and what search behavior will be.

Here is a simple search form:

```
<FORM ACTION="[area search]" METHOD=POST>
<INPUT TYPE="text" SIZE="30" NAME="mv_searchspec">
<INPUT TYPE="submit" VALUE="Search">
</FORM>
```

When the "Search" submit button is pressed (or <ENTER> is pressed), Interchange will search the `products.txt` file for the string entered into the text field `mv_searchspec`, and return the product code pertaining to that line.

The same search for a fixed string, say "shirt," could be performed with the use of a hot link, using the special scan URL:

```
[page search="se=shirt"]See our shirt collection!</a>
```

The default is to search every field on the line. To match on the string "shirt" in the product database field "description," modify the search:

```
<INPUT TYPE="hidden" NAME="mv_search_field" VALUE="description">
```

In the hot−linked URL search:

```
[page search="
              se=shirt
              sf=category
          "]See our shirt collection!</a>
```

To let the user decide on the search parameters, use checkboxes or radiobox fields to set the fields:

```
Search by author
   <INPUT TYPE="checkbox" NAME="mv_search_field" VALUE="author">
Search by title
    <INPUT TYPE="checkbox" NAME="mv_search_field" VALUE="title">
```

Fields can be stacked. If more than one is checked, all checked fields will be searched.

5.2. Glimpse

To use the Glimpse search, the Glimpse index must be built based on files in the ProductDir, or wherever the files to be searched will be located. If the catalog is in `/var/lib/interchange/foundation`, the command line to build the index for the products file would be:

```
chdir /var/lib/interchange/foundation/products
glimpseindex −b −H . products.txt
```

There are several ways to improve search speed for large catalogs. One method that works well for large `products.txt` files is to split the `products.txt` file into small index files (in the example, 100 lines) with the split(1) UNIX/POSIX command. Then, index it with Glimpse:

```
split −100 products.txt index.txt.
glimpseindex −H /var/lib/interchange/foundation/products index.txt.*
```

This will dramatically increase search speeds for large catalogs, at least if the search term is relatively unique. If it is a common string, in a category search, for example, it is better to use the text−based search.

To search for numbers, add the `−n` option to the Glimpse command line.

Note: A large catalog is one of more than several thousand items; smaller ones have acceptable speed in any of the search modes.

If the Glimpse executable is not found at Interchange startup, the Glimpse search will be disabled and the regular text−based search used instead.

There are several things to watch for while using Glimpse, and a liberal dose of the Glimpse documentation is suggested. In particular, the spelling error capability will not work in combination with the field−specific search. Glimpse selects the line, but Interchange's text−based search routines disqualify it when checking to see if the search string is within one of the specified fields.

To use field−specific searching on Glimpse, tell it what the field names are. If the search is on the products database (file), nothing is needed for the default is to use the field names from the products database. If it is

some other field layout, specify the file to get the field names from with `mv_field_file` (ff).

5.3. Fast Binary Search

Fast binary searching is useful for scanning large databases for strings that match the beginning of a line. They use the standard Perl module Search::Dict, and are enabled through use of the `mv_dict_look`, `mv_dict_end`, `mv_dict_limit`, `mv_dict_fold`, and `mv_dict_order` variables.

The field to search is the first field in the file, the product code should be in the second field, delimited by TAB. Set the `mv_return_fields=1` to return the product code in the search.

The search must be done on a dictionary−ordered pre−built index, which can be produced with the database INDEX modifier. See Dictionary indexing with INDEX.

If using the `mv_dict_look` parameter by itself, and the proper index file is present, Interchange will set the options:

```
mv_return_fields=1
mv_dict_limit=-1
```

This will make the search behave much like the simple search described above, except it will be much faster on large files and will match only from the beginning of the field. Here is an example. A `title` index has been built by including in `catalog.cfg`:

```
Database   products   INDEX    title
```

Note: The ASCII source file must be "touched" to rebuild the index and the database.

Now, specify in a form:

```
<FORM ACTION="[process href=search]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_dict_limit VALUE=title>
<INPUT NAME=mv_dict_look>
</FORM>
```

or in a URL:

```
[page search="dl=Van Gogh/di=title"]
```

This search is case−sensitive. To do the same thing case−insensitively:

```
Database   products   INDEX    title:f

<FORM ACTION="[process href=search]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_dict_limit VALUE=title>
<INPUT TYPE=hidden NAME=mv_dict_fold  VALUE=1>
<INPUT NAME=mv_dict_look>
</FORM>

[page search="dl=Van Gogh/di=title/df=1"]
```

5.4. Coordinated and Joined Searching

Interchange will do a complete range of tests on individual columns in the database. To use this function, set `mv_coordinate` to Yes (co=yes in the one−click syntax). In order to use coordinated searching, the number of search fields must equal the number of search strings.

To make sure that is the case, use the `mv_search_map` variable. It allows variables to be mapped to others in the search specification. For example:

```
<INPUT TYPE=hidden NAME=mv_search_map VALUE="
    mv_searchspec=search1
    mv_searchspec=search2
    mv_searchspec=search3
    ">
<INPUT TYPE=hidden NAME=mv_search_field VALUE=title>
<INPUT TYPE=hidden NAME=mv_search_field VALUE=artist>
<INPUT TYPE=hidden NAME=mv_search_field VALUE=category>
Artist: <INPUT NAME=search1 VALUE="">
Title:  <INPUT NAME=search2 VALUE="">
Genre:  <INPUT NAME=search3 VALUE="">
```

Even if the user leaves one blank, the search will work.

Leading/trailing whitespace is stripped from all lines in the `mv_search_map` variable, so it can be positioned as shown for convenience.

Coordinated searches may be joined with the output of another table if set one of the `mv_search_field` values is set to a `table:column` pair. Note that this will slow down large searches considerably unless there is another search specification, as the database must be accessed for every search line If there is a search field that qualifies for a regular expression search function, or conducting a binary search with `mv_dict_look`, or are not doing an `OR` search, the penalty should not be too great as only matching lines will cause an access to the database.

Individual field operations can then be specified with the `mv_column_op` (or op) parameter. The operations include:

```
operation            string     numeric   equivalent
---------
equal to               eq         ==          =
not equal              ne         !=          <>
greater than           gt         >
less than              lt         <
less than/equal to     le         <=
greater than/equal to  ge         >=
regular expression     rm                      =~ , LIKE
regular expression NOT rn                      !~
exact match            em
Text::Query::Advanced  aq
Text::Query::Simple    tq
```

An example:

```
[page search="
        co=yes
        sf=title
```

```
        se=Sunflowers
        op=em
        sf=artist
        se=Van Gogh
        op=rm
"] Sunflowers, Van Gogh </a>

[page search="
        co=yes

        sf=title
        se=Sunflowers
        nu=0
        op=!~

        sf=artist
        se=Van Gogh
        op=rm
        nu=0

        sf=inventory:qty
        se=1
        op=>=
        nu=1
"] Any in stock except Sunflowers, Van Gogh </a>
```

Note that in the second example, nu=0 must be specified even though that is the default. This is to set the proper correspondence. To avoid having to do this, use Interchange's option array feature:

```
[page search.0="
                sf=title
                se=Sunflowers
                op=!~
            "
      search.1="
                sf=artist
                se=Van Gogh
            "
      search.2="
                sf=inventory:qty
                se=1
                op=>=
                nu=1
            "
    ] Any in stock except Sunflowers, Van Gogh </a>
```

The `co=yes` is assumed when specifying a multiple search.

The second search will check the stock status of the painting provided there is an `inventory` table as in some of the Interchange demo catalogs. If the `qty` field is greater than or equal to 1, the product will be picked. If out of stock, it will not be found.

It always helps to have an `rm` type included in the search. This is used to pre−screen records so that database accesses only need be made for already−matching entries. If accesses must be made for every record, large searches can get quite slow.

The special `aq` and `tq` query types only operate if the `Text::Query` CPAN module is installed. This allows Altavista−style searches on the field, using AND, OR, NOT, and NEAR with arbitrarily complex

parentheses.

A useful form for the `aq` type would be:

```
<form action="[area search]" method=POST>
<input type=hidden name=mv_session_id value="[data session id]">
<input type=hidden name=mv_column_op VALUE="aq">
<input type=hidden name=mv_coordinate VALUE=1>
<input type=hidden name=mv_min_string value=2>
<input type=hidden name=mv_search_field VALUE=":sku:description:comment:category">
<input type=hidden name=mv_searchtype VALUE=db>
<input name=mv_searchspec type=text size=12>
<input type=submit value="SEARCH">
</form>
```

This searches the sku, description, comment, and category fields in the default products file with Text::Query syntax. Try the term "painters NEAR set" in the default foundation example.

5.5. Custom search operators

You can write your own search operator with Interchange's CodeDef. In `interchange.cfg`, or in the `code` directory tree, you can put:

```
CodeDef find_mirrored SearchOp
CodeDef find_mirrored Routine <<EOR
sub {
      my ($self, $i, $pat) = @_;
      $pat = reverse $pat;
      return sub {
              my $string = shift;
              $string =~ /$pat/io;
      };
}
EOR
```

Now you can do:

```
[loop search="
            se=sretniap
            sf=description
            fi=products
            st=db
            co=yes
            rf=*
            op=find_mirrored
        "]
[loop-code] [loop-param description]<br>
[/loop]
```

The passed parameters are:

- The search object ($self)
- The index into coordinated search array ($i)
- The pattern to match
- The name of the op (find_hammer in this case)

Must return a sub which receives the data to match and returns 1 if it matches. DOES NOT HONOR mv_negate UNLESS you tell it to.

See Vend::Search::create_text_query for an example of how to return a proper routine and look in search object for the associated params.

5.6. Specifying a Text−Based Search with SQL Syntax

If the Perl `SQL::Statement` module is installed, SQL syntax can be specified for the text−based search. This is not the same as the external SQL database search, treated below separately. This works on the ASCII text source file, not on the actual database.

This syntax allows this form setup:

```
Artist: <INPUT NAME="artist">
Title:  <INPUT NAME="title">
<INPUT TYPE=hidden NAME="mv_sql_query"
        VALUE="
            SELECT code FROM products
            WHERE artist LIKE artist
            AND    title LIKE title">
```

If the right hand side of an expression looks like a column, i.e., is not quoted, the appropriate form variable is substituted. (If used in a one−click, the corresponding scratch variable is used instead.) The assumption is reversed for the left−hand side. If it is a quoted string, the column name is read from the passed values. Otherwise, the column name is literal.

```
Search for: <INPUT NAME="searchstring"><BR>
Search in   <INPUT TYPE="radio" NAME="column" VALUE="title"> title
    <INPUT TYPE="radio" NAME="column" VALUE="artist"> artist
    <INPUT TYPE=hidden NAME="mv_sql_query"
      VALUE="SELECT code FROM products WHERE 'column' LIKE searchstring">
```

Once again, this does not conduct a search on an SQL database, but formats a corresponding text−based search. Parentheses will have no effect, and an OR condition will cause all conditions to be OR. The searches above would be similar to:

```
[page search="
            co=yes
            sf=artist
            op=rm
            se=[value artist]
            sf=title
            op=rm
            se=[value title]
        "  ]
    Search for [value artist], [value title]
</a>

[page search="
            co=yes
            sf=[value column]
            op=rm
            se=[value searchstring]
        "  ]
Search for [value searchstring]
```

```
        in  [value column]
  </a>
```

5.7. One−Click Searches

Interchange allows a search to be passed in a URL, as shown above. Just specify the search with the special page parameter search or special page `scan`. Here is an example:

```
[page search="
           se=Impressionists
           sf=category
       "]
   Impressionist Paintings
</a>
```

This is the same:

```
[page scan se=Impressionists/sf=category]
   Impressionist Paintings
</a>
```

Here is the same thing from a home page (assuming /cgi−bin/vlink is the CGI path for Interchange's vlink):

```
<A HREF="/cgi−bin/vlink/scan/se=Impressionists/sf=category">
   Impressionist Paintings
</A>
```

The two−letter abbreviations are mapped with these letters:

```
ac  mv_all_chars
bd  mv_base_directory
bs  mv_begin_string
ck  mv_cache_key
co  mv_coordinate
cs  mv_case
cv  mv_verbatim_columns
de  mv_dict_end
df  mv_dict_fold
di  mv_dict_limit
dl  mv_dict_look
DL  mv_raw_dict_look
do  mv_dict_order
dr  mv_record_delim
em  mv_exact_match
er  mv_spelling_errors
ff  mv_field_file
fi  mv_search_file
fm  mv_first_match
fn  mv_field_names
hs  mv_head_skip
ix  mv_index_delim
lb  mv_search_label
lf  mv_like_field
lo  mv_list_only
lr  mv_search_line_return
ls  mv_like_spec
ma  mv_more_alpha
mc  mv_more_alpha_chars
```

```
md  mv_more_decade
ml  mv_matchlimit
mm  mv_max_matches
MM  mv_more_matches
mp  mv_profile
ms  mv_min_string
ne  mv_negate
ng  mv_negate
np  mv_nextpage
nu  mv_numeric
op  mv_column_op
os  mv_orsearch
pf  prefix
ra  mv_return_all
rd  mv_return_delim
rf  mv_return_fields
rn  mv_return_file_name
rr  mv_return_reference
rs  mv_return_spec
se  mv_searchspec
sf  mv_search_field
sg  mv_search_group
si  mv_search_immediate
sm  mv_start_match
sp  mv_search_page
sq  mv_sql_query
sr  mv_search_relate
st  mv_searchtype
su  mv_substring_match
tf  mv_sort_field
to  mv_sort_option
un  mv_unique
va  mv_value
```

These can be treated just the same as form variables on the page, except that they can't contain a new line. If using the multi−line method of specification, the characters will automatically be escaped for a URL.

IMPORTANT NOTE: An incompatibility in earlier Interchange catalogs is specifying `[page scan/se=searchstring]`. This is interpreted by the parser as `[page scan/se="searchstring"]` and will cause a bad URL. Change this to `[page scan se=searchstring]`, or perhaps better yet:

```
[page search="
                se=searchstring
      "]
```

A one−click search may be specified in three different ways.

Original

To do an OR search on the fields category and artist for the strings "Surreal" and "Gogh," while matching substrings, do:

```
[page scan se=Surreal/se=Gogh/os=yes/su=yes/sf=artist/sf=category]
   Van Gogh −− compare to surrealists
</a>
```

In this method of specification, to replace a / (slash) in a file name (for the sp, bd, or fi parameter), the shorthand of :: must be used, i.e., sp=results::standard. (This may not work for some browsers, so put the page in the main pages directory or define the page in a search profile.)

Multi−Line

Specify parameters one to a line, as well.

```
[page scan
    se="Van Gogh"
    sp=lists/surreal
    os=yes
    su=yes
    sf=artist
    sf=category
] Van Gogh −− compare to surrealists </a>
```

Any "unsafe" characters will be escaped. To search for trailing spaces (unlikely), quote.

Ampersand

Substitute & for / in the specification and be able to use / and quotes and spaces in the specification.

```
[page href=scan se="Van Gogh"&sp=lists/surreal&os=yes&su=yes&sf=artist]
   Van Gogh −− compare to surrealists
</a>
```

Any "unsafe" characters will be escaped.

5.8. Setting Display Options with mv_value

A value can be specified that will be set in the link with the `mv_value` parameter. It takes an argument of `var=value`, just as setting a normal variable in an Interchange profile. Actually `mv_value` is a misnomer, it will almost never be used in a form where variable values can be set. Always specify it in a one−click search with `va=var=value`. Example:

```
[page href=scan
      arg="se=Renaissance
           se=Impressionists
           va=category_name=Renaissance and Impressionist Paintings
           os=yes"]Renaissance and Impressionist Paintings</a>
```

Display the appropriate category on the search results page with `[value category_name]`.

5.9. In−Page Searches

To specify a search inside a page with the `[search−region parameters*]` tag. The parameters are the same as the one−click search, and the output is always a newline−separated list of the return objects, by default, a series of item codes.

The `[loop ...]` tag directly accepts a search parameter. To search for all products in the categories "Americana" and "Contemporary," do:

```
[loop search="
    se=Americana
    se=Contemporary
    os=yes
    sf=category9
    "]
Artist: [loop-field artist]<BR>
Title: [loop-field title]<P>
[/loop]
```

The advantage of the in-page search is that searches can be embedded within searches, and there can be straight unchanging links from static HTML pages.

To place an in-page search with the full range of display in a normal results page, use the `[search-region]` tag the same as above, except that `[search-list]`, `[more-list]`, and `[more]` tags can be placed within it. Use them to display and format the results, including paging. For example:

```
[search-region  more=1
                search="
                     se=Americana
                     sf=category
                     ml=2
                "]
[more-list][more][/more-list]
[search-list]
[page [item-code]]
    [item-field title]<A>, by [item-field artist]
[/search-list]
[no-match]
    Sorry, no matches for [value mv_searchspec].
[/no-match]
[/search-region]
```

Note: The [item-code] above does not need to be quoted because it is replaced before the [page ...] tag is interpolated. If building large lists, this is worth doing because unquoted tags are twice as fast to parse.

To use the same page for search paging, make sure to set the `sp=page` parameter.

5.10. Search Profiles

An unlimited number of search profiles can be predefined that reside in a file or files. To use this, make up a series of lines like:

```
mv_search_field=artist
mv_search_field=category
mv_orsearch=yes
```

These correspond to the Interchange search variables that can be set on a form. Set it right on the page that contains the search.

```
[set artist_profile]
mv_search_field=artist
mv_search_field=category
mv_orsearch=yes
[/set]
```

This is the same:

```
[set artist_profile]
sf=artist
sf=category
os=yes
[/set]
```

Then, in the search form, set a variable with the name of the profile:

```
   <INPUT TYPE=hidden NAME=mv_profile VALUE=artist_profile>
```

In a one−click search, use the `mp` modifier:

```
[page scan se=Leonardo/mp=artist_profile]A left−handed artist</a>
```

They can also be placed in a file. Define the file name in the `SearchProfile` directive. The catalog must be reconfigured for Interchange to read it. The profile is named by placing a name following a __NAME__ pragma:

```
 __NAME__ title_search
```

The __NAME__ must begin the line, and be followed by whitespace and the name.

The special variable `mv_last` stops interpretation of search variables. The following variables are always interpreted:

```
   mv_dict_look
   mv_searchspec
```

Other than that, if `mv_last` is set in a search profile, and there are other variables on the search form, they will not be interpreted.

To place multiple search profiles in the same file, separate them with __END__, which must be on a line by itself.

5.11. Search Reference

The supplied `simple/srchform.html` and `simple/results.html` pages show example search forms. Modify them to present the search in any way desired. Be careful to use the proper variable names for passing to Interchange. It is also necessary to copy the hidden variables as−is. They are required to interpret the request as a search.

Note: The following definitions frequently refer to field name and column and column number. All are the references to the columns of a searched text file as separated by delimiter characters.

The field names can be specified in several ways.

ProductFiles

If the file to be searched is left empty in the search form or definition (it is set with `mv_search_file (fi)`), the text files associated with the products databases will be searched, and field names are already

available as named in the first line of the file(s). This is defined to be `products.txt` in the Interchange demo catalogs.
Be careful if using SQL! If the database is changed and not exported with `[tag export products][/tag]`, searches will not be successful.

Other database files

If the file or files to be searched are ASCII delimited files, and have field names specified on the first line of the file, Interchange will read the first line (of the first file) and determine the field names.

Other files

If the file or files to be searched are ASCII delimited files, but don't have field names specified on the first line of the file, set the variable `mv_field_names` to a comma−separated list of field names as they will be referenced.

Fields can also always be specified by an integer column number, with 0 as the first column.

mv_all_chars

Scan abbreviation: ac=[1|0]. Set this if searching is anticipated for lots of punctuation characters that might be special characters for Perl. The characters ()[]\$^ are included.

mv_base_directory

Scan abbreviation: bd=/directory/name. In the text search, set to the directory from which to base file searches. File names without leading / characters will be based from there. In the Glimpse search, passed to Glimpse with the `−H` option, and Glimpse will look for its indices there. Default is ProductDir.
If an absolute path directory is used, for security enable it in the users session with:

```
[set /directory/name]1[/set]
```

This prevents users from setting an arbitrary value and viewing arbitrary files.

mv_begin_string

If this is set, the string will only match if it is at the beginning of a field. The handling is a bit different for the default AND search compared to the OR search. With OR searches all words are searched for from the beginning of the field, with AND searches all are.
This is a multiple parameter. If `mv_coordinate` is in force, it should be set as many times as necessary to match the field/searchstring combination. If set only once, it applies to all fields. If set more than once but not as many times as the fields, it will default to off.

mv_cache_key

Not normally set by the user. It is a value that provides a pointer to the search reference by the `more` function.

mv_case

If this item is set to `No`, the search will return items without regard to upper or lower case. This is the default. Set to `Yes` if case should be matched. Implement with a checkbox <INPUT TYPE=CHECKBOX> field.

If stacked to match the `mv_search_field` and `mv_searchspec` variables, and `mv_coordinate` is set, it will operate only for the corresponding field.
Scan abbreviation: cs

mv_column_op

In the coordinated search, the operation that will be performed to check the field for a search match. These operations are supported:

```
!=              Not equal to
!~              Not matching regular expression
<               Less than
<=              Less than or equal to
<>              Not equal to
=               Equal to
==              Equal to
=~              Matching regular expression
>               Greater than
>=              Greater than or equal to
em              Exact match
eq              Equal to
ge              Greater than or equal to
gt              Greater than
le              Less than or equal to
lt              Less than
ne              Not equal to
rm              Matching regular expression
rn              Not matching regular expression
```

If stacked to match the `mv_search_field` and `mv_searchspec` variables, and `mv_coordinate` is set, it will operate only for the corresponding field.

Note that several of the operators are the same. They do either numeric or string comparisons based on the status of `mv_numeric` (alias `nu`) for that column.

mv_coordinate

> If this item is set to `Yes`, and the number of search fields equals the number of search specs, the search will return only items that match field to spec. (The search specifications are set by stacked `mv_searchspec` and `mv_search_field` variables.)
> Case sensitivity, substring matching, and negation all work on a field−by field basis according to the following:

If only one instance of the option is set, it will affect all fields.
If the number of instances of the option is greater than or equal to the number of search specs, all will be used independently. Trailing instances will be ignored.
If more than one instance of the options are set, but fewer than the number of search specifications, the default setting will be used for the trailing unset options.
If a search specification is blank, it will be removed and all case−sensitivity/negation/substring options will be adjusted accordingly. If you need a blank string to match on, use quotes (`""`).

mv_dict_end

If the string at the beginning of a line lexically exceeds this value, matching will stop. Ignored without

mv_dict_look.

mv_dict_fold

Make dictionary matching case−insensitive. Ignored without mv_dict_look.

Note: This is the reverse sense from mv_case.

---

mv_dict_limit

Automatically set the limiting string (mv_dict_end) to be one character greater than the mv_dict_look variable, at the character position specified. A value of 1, for instance, will set the limiting string to "fprsythe" if the value of mv_dict_look is "forsythe". A useful value is −1, which will increment the last character (setting the mv_dict_end to "forsythf" in our example). This prevents having to scan the whole file once a unique match is found.

Note: The order of this and the mv_dict_end variable is significant. Each will overwrite the other.

If this is set to a non−numeric value, an automatic mode is entered which looks for a dictionary−indexed file that corresponds to the file name plus .field, where field is whatever mv_dict_limit is set to. The actual value of mv_dict_limit is set to −1. If the file does not exist, the original file is silently used. Also, the value of mv_return_fields is set to 1 to correspond to the location of the key in the auto−indexed file. To illustrate:

```
<INPUT TYPE=hidden NAME=mv_dict_limit  VALUE=category>
<INPUT TYPE=hidden NAME=mv_search_file VALUE="products.txt">
```

is equal to:

```
<INPUT TYPE=hidden NAME=mv_dict_limit    VALUE="−1">
<INPUT TYPE=hidden NAME=mv_search_file   VALUE="products.txt.category">
<INPUT TYPE=hidden NAME=mv_return_fields VALUE="1">
```

The real utility would be in a form construct like

```
Search for
<SELECT NAME=mv_dict_limit>
<OPTION> author
<OPTION> title
</SELECT> beginning with <INPUT NAME=mv_dictlook>
```

which would allow automatic binary search file selection.
Combined with the INDEX attribute to the Database directive, this allows fast binary search qualification combined with regular mv_searchspec text searches.

mv_dict_look

The string at which to begin matching at in a dictionary−based search. If not set, the mv_dict_end, mv_dict_fold, and mv_dict_case variables will be ignored. May be set in a search profile based on other form variables.

mv_dict_order

Make dictionary matching follow dictionary order, where only word characters and whitespace matter. Ignored without `mv_dict_look`.

mv_doit

This can be set to `search` to make a form with a `process` action be a search page by default. The `mv_todo` variable takes precedence.

mv_exact_match

Normally Interchange searches match words, as opposed to sentences. This behavior can be overridden with `mv_exact_match`, which when set will place quotes around any value in `mv_searchspec` or `mv_dict_look`.

mv_field_file

If you want to search a file which has no field header on the first line, you can specify a file to get the field names from. It expects a single line with the field names separated by TAB characters.

mv_field_names

Deprecated in favor of in−list sorting. Defines the field names for the file being searched. This guarantees that they will be available, and prevents a disk access if using named fields on a search file (that is not the product database ASCII source, where field names are already known). This must be exactly correct, or it will result in anomalous search operation. Usually passed in a hidden field or search profile as a comma−separated list.

Note: Use this on the product database only if planning on both pre−sorting with `mv_sort_field` and then post−sorting `with [sort]field:opt[/sort]`.

---

mv_first_match

Normally Interchange will return the first page of a search. If this variable is set, it will start the search return at the match specified, even if there is only one page. If set to a value greater than the number of matches, it will act as if no matches were found.

mv_head_skip

Normally Interchange searches all lines of an index/product file but the first. Set this to the number of lines to skip at the beginning of the index. Default is 1 for the text search, which skips the header line in the product file. Default is 0 for a Glimpse search.

mv_index_delim

Sets the delimiter for counting fields in a search index. The default is TAB. It should rarely be changed unless you are searching a pipe−delimited or colon−delimited file.

mv_like_field

Specifies a field in a database search which should be used for a screening function based on the SQL `like` function. Needs `mv_like_spec`.

mv_like_spec

The string that should be searched for in `mv_like_field`. The behavior of the % character and case−sensitivity depends on your SQL implementation.

mv_matchlimit

Function depends upon context. When the search results display is handled by one of the mechanisms which works with `[more]` lists (such as `[search−region]`), mv_matchlimit determines the number of results per page. If more matches than mv_matchlimit are found, the search paging mechanism will be employed if the proper `[more−list]` is present. When the search results are displayed as one continuous list (i.e.: with `[loop search="..."]`), mv_matchlimit is equivalent in function to mv_max_matches.
To have no matchlimit, use **none** instead of a number. **all** does the same thing (since returning "all" is just anothing way of looking at no matchlimit).
If no matchlimit is provided, or an invalid setting (some other string or 0) the default is taken from catalog variable MV_DEFAULT_MATCHLIMIT, and if that's not set, is 50.

mv_max_matches

The maximum number of records that will be returned in a search. Default is unlimited. If search results paging with `[more−list]` is to be employed, Use `mv_matchlimit` to set the number of results per page.

mv_min_string

Sets the minimum size of a search string for a search operation. Default is 4 for the Glimpse search, and 1 for the text search.

mv_negate

Specifies that records NOT matching the search criteria will be returned. Default is no. It is not operative for the Glimpse search.
If stacked to match the `mv_search_field` and `mv_searchspec` variables, and `mv_coordinate` is set, it will operate only for the corresponding field.

mv_orsearch

If this item is set to `Yes`, the search will return items matching any of the words in `searchspec`. The default is No.

mv_profile

Selects one of the pre−defined search specifications set by the `SearchProfile` directive. If the special variable within that file, `mv_last`, is defined, it will prevent the scanning of the form input for further search modifications. The values of `mv_searchspec` and `mv_dict_look` are always scanned, so specify this to do the equivalent of setting multiple checkboxes or radioboxes with one click, while still reading the search input text.

mv_record_delim

Sets the delimiter for counting records in a search index. The default is newline, which works for the products and most line−based index files.

mv_return_fields

The field(s) that should be returned by the match, specified either by field name or by column number, separated by commas. Do not list the same field more than once per search. Specify 0 as the first field to be returned if searching the products database, since that is the key for accessing database fields.

As with SQL queries, you can use the '*' shortcut to return all fields. For example:

```
[loop search="fi=nation/ra=yes/rf=*"]
```

when used with a hypothetical 'nation' table would be equivalent to:

```
[loop search="
      fi=nation
      ra=yes
      rf=code,sorder,region,name,tax
"]
```

as well as:

```
[loop search="fi=nation/ra=yes/rf=0,1,2,3,4"]
```

and:

```
[query sql="select * from nation"][/query]
```

However, you probably rarely need to use every single field in a row. For maximum maintainability and execution speed the best practice is to list by name only the fields you want returned.

mv_return_spec

Returns the string specified as the search (i.e., the value of `mv_searchspec`) as the one and only match. Typically used in a SKU/part number search.

mv_search_field

The field(s) to be searched, specified either by column name or by column number.
If the number of instances matches the number of fields specified in the `mv_searchspec` variable and `mv_coordinate` is set to true, each search field (in order specified on the form) will be matched with each search spec (again in that order).

mv_search_file

In the text search, set this variable to the file(s) to be scanned for a match. The default, if not set, is to scan the default ProductFiles (i.e., products.txt). If set multiple times in a form (for a text search), will cause a search all the files. One file name per instance.
In the Glimpse search, follows the Glimpse wildcard−based file name matching scheme. Use with caution and a liberal dose of the Glimpse man page.

mv_search_match_count

Set by the search to indicate the total number of matches found.

mv_search_page

The Interchange−style name of the page that should display the search results. This overrides the default value of `search`.

mv_searchspec

The actual search string that is typed in by the customer. It is a text INPUT TYPE=TEXT field, or can be put in a select (drop−down) list to enable category searches. If multiple instances are found, they will be concatenated just as if multiple words had been placed in a text field.
The user can place quotes around words to specify that they match as a string. To enable this by default, use the `mv_exact_match` variable.
If `mv_dict_look` has a value, and `mv_searchspec` does not, then `mv_searchspec` will be set to the value of `mv_dict_look`.
If the number of instances matches the number of fields specified in the `mv_search_field` variable and `mv_coordinate` is set to true, each search field (in order specified on the form) will be matched with each search spec (again in that order).

mv_searchtype

If set to Glimpse, selects the Glimpse search (if Glimpse is defined).
If set to db, iterates over every row of the database (not the associated text source file).
If set to sql, same as `db`.
If set to text, selects the text−based search.
When using `st=db`, returned keys may be affected by `TableRestrict`. See `CATALOG.CFG`.
Defaults to text if Glimpse is not defined; defaults to Glimpse if it is defined. This can allow use of both search types if that is desirable. For instance, searching for very common strings is better done by the text−based search. An example might be searching for categories of items instead of individual items.

mv_small_data

Tells the search engine that there is a small amount of data in the file and that it should perform the search function on every line.

Normally, when Interchange can find a fixed search expression it produces a "screening" function which will allow records to be quickly rejected when they don't match. If there are less than 50 records in the file or database, this may be counterproductive.

mv_sort_field

The file field(s) the search is to be sorted on, specified in one of two ways. If the file(s) to be searched have a header line (the first line) that contains delimiter−separated field names, it can be specified by field name. It can also be specified by column number (the code or key is specified with a value of 0, for both types). These can be stacked if coming from a form or placed in a single specification separated by commas.

Note: If specifying a sort for the product database, `mv_field_names` must be specified if doing a fieldname−addressed post−sort.

mv_sort_option

The way that each field should be sorted. The flags are `r`, `n`, and `f`, reverse, numeric, and case−insensitive respectively. These can be stacked if coming from a form or placed in a single specification separated by commas. The stacked options will be applied to the sort fields as they are defined, presuming those are stacked.

mv_spelling_errors

The number of spelling errors that will be tolerated. Ignored unless using Glimpse. For a large table, limit this to two.

mv_substring_match

If `mv_substring_match` is set to `Yes`, matches on substrings as well as whole words. Typically set this for dictionary−based searches.
If stacked to match the `mv_search_field` and `mv_searchspec` variables and `mv_coordinate` is set, it will operate only for the corresponding field.

mv_unique

If set to a true value, causes the sort to return only unique results. This operates on whatever the search return is, as defined by `mv_return_fields`.

mv_value

This is normally only used in the one−click search (va=var=value). It allows setting of a session variable based on the clicked link, which makes for easy definition of headers and other display choices. (If had trouble using `mv_searchspec` for this before, this is what is needed.)

5.12. The Results Page

Once a search has been completed, there needs to be a way of presenting the output. By default, the `SpecialPage` search is used. It is set to `results` in the distribution demo, but any number of search pages can be specified by passing the value in the search form specified in the variable `mv_search_page`.

On the search page, some special Interchange tags are used to format the otherwise standard HTML. Each of the iterative tags is applied to every code returned from the search. This is normally the product code, but could be a key to any of the arbitrary databases. The value placed by the `[item−code]` tag is set to the first field returned from the search.

The basic structure looks like this:

```
[search−region]
[search−list]
    your iterating code, once for each match
[/search−list]
[no−match]
    Text / tags to be output if no matches found (optional but recommended)
[/no−match]
```

```
[more-list]
    More / paging area (optional)
[/more-list]
[/search-region]
```

Tip for catalogs upgraded from Minivend 3: A `[search-list][/search-list]` must always be surrounded by a `[search-region][/search-region]` pair. This is a change from Minivend 3.

[search-list]

Starts the representation of a search list. Interchange tags can be embedded in the search list, yielding a table or formatted list of items with part number, description, price, and hyperlinks to order or go to its catalog page.
The example tags shown have an `item-` prefix, which is the default. Set any prefix desired with the `prefix` parameter to `[search-region]`:

```
[search-region prefix=my]
[search-list]
    SKU:   [my-code]
    Title: [my-data products title]
[/search-list]
[/search-region]
```

The standard set of Interchange iterative ITL tags are available. They are interpolated in this order:

```
[item-alternate N] true [else] false [/else] [/item-alternate]
[if-item-param named_field] true [else] false [/else] [/if-item-param]
[item-param named_field]
[if-item-pos N] true [else] false [/else] [/if-item-pos]
[item-pos N]
[if-item-field products_field] true [else] false [/else] [/if-item-field]
[item-field products_column]
[item-increment]
[item-accessories]
[item-code]
[item-description]
[if-item-data table column] true [else] false [/else] [/if-item-data]
[item-data table column]
[item-price N* noformat=1*]
[item-calc] [/item-calc]
[item-change marker]
       [condition]variable text[/condition]
       true
       [else] false [/else]
[/item-change marker]
[item-last] condition [/item-last]
[item-next] condition [/item-next]
```

Note: those that reference the shopping cart do not apply, i.e., [item-quantity], [item-modifier ...] and friends.

[/search-list]

Ends the search list.

[no-match]

Starts the region of the search results page that should be returned if there is no match (and no error) for the search. If this is not on the page, the special page nomatch will be displayed instead.

[/no−match]

Ends the no match region.

[sort database:field:option* database:field:option*]

Sorts the search list return based on database fields. If no options are supplied, sorts according to the return code. See SORTING.
This is slow, and it is far better to pre−sort the return in the search specification.

[item−change marker]

Active only within `[search−list][/search−list]`.
Along with the companion `[/item−change marker]`, surrounds a region which should only be output when a field (or other repeating value) changes its value. This allows indented lists similar to database reports to be easily formatted. The repeating value must be a tag interpolated in the search process, such as `[item−field field]` or `[item−data database field]`.
Of course, this will only work as expected when the search results are properly sorted.
The `marker` field is mandatory, and is also arbitrary, meaning that any marker can be selected as long as it matches the marker associated with `[/item−change marker]`. The value to be tested is contained within a `[condition]value[/condition]` tag pair. The `[item−change marker]` tag also processes an `[else] [/else]` pair for output when the value does not change. The tags may be nested as long as the markers are different.
The following is a simple example for a search list that has a field `category` and `subcategory` associated with each item:

```
<TABLE>
<TR><TH>Category</TH><TH>Subcategory</TH><TH>Product</TH></TR>
[search−list]
<TR>
   <TD>
        [item−change cat]

        [condition][item−field category][/condition]

                [item−field category]
        [else]
                 
        [/else]
        [/item−change cat]
   </TD>
   <TD>
        [item−change subcat]

        [condition][item−field subcategory][/condition]

                [item−field subcategory]
        [else]
                 
        [/else]
        [/item−change subcat]
   </TD>
   <TD> [item−field name] </TD>
```

```
[/search-list]
</TABLE>
```

The above should output a table that only shows the category and subcategory once, while showing the name for every product. (The ` ` will prevent blanked table cells if using a border.)

[/item-change marker]

Companion to `[item-change marker]`.

[matches]

Replaced with the range of match numbers displayed by the search page. Looks something like "1-50". Make sure to insert this item between a `[more-list]` and `[/more-list]` element pair.

[match-count]

Replaced with the total number of matches. This tag works even on `[query]` searches where `[value mv_search_match_count]` isn't set unless the query is applied to a non-SQL database. Make sure to insert this item between a `[more-list]` and `[/more-list]` element pair.

[more-list next_img* prev_img* page_img* border* border_current*]

Starts the section of the search page which is only displayed if there are more matches than specified in `mv_matchlimit`. If there are less matches than the number in mv_matchlimit, all text/html between the `[more_list]` and `[/more_list]` elements is stripped.
Use in conjunction with the `[more]` element to place pointers to additional pages of matches.
If the optional arguments `next_img`, `prev_img`, and/or `page_img` are present, they represent image files that will be inserted instead of the standard 'Next,' 'Previous,' and page number. If `prev_img` is `none`, then no previous link will be output. If `page_img` is `none`, then no links to pages of matches will be output. These are URLs, are substituted for with *ImageDir* and friends, and will be encased in IMG tags. Lastly, `border` is the border number to put.
In addition, if `page_img` is used, it will be passed an argument of the digit that is to be represented. This would allow an image generator program to be used, generating page numbers on the fly. The `border` and `border_selected` values are integers indicating the border that should be put around images in the `page_img` selection. The <border_selected> is used for the current page if set.
\Examples:
`[more-list next.gif prev.gif page_num.cgi 3]` causes anchors of:

```
Previous   <IMG SRC="prev.gif" Border=3>
Page 1     <IMG SRC="/cgi-bin/page_num.cgi?1">
Page 2     <IMG SRC="/cgi-bin/page_num.cgi?2">
Next       <IMG SRC="next.gif" Border=3>
```

`[more-list next.gif prev.gif page_num.cgi]` causes anchors of:

```
Previous   <IMG SRC="prev.gif">
Page 1     <IMG SRC="/cgi-bin/page_num.cgi?1">
Page 2     <IMG SRC="/cgi-bin/page_num.cgi?2">
Next       <IMG SRC="next.gif">
```

`[more-list next.gif prev.gif 0 0]` causes anchors of:

```
Previous   <IMG SRC="prev.gif" Border=0>
Page 1     <IMG SRC="/cgi-bin/page_num.cgi?1">
Page 2     <IMG SRC="/cgi-bin/page_num.cgi?2">
Next       <IMG SRC="next.gif" Border=0>
```

To set custom text for the "Previous" and "Next" usually used, define the `next_img`, `prev_img`, and `page_img` with `[next-anchor][/next-anchor]`, `[prev-anchor][/prev-anchor]`, `[first-anchor][/first-anchor]`, `[last-anchor][/last-anchor]` and `[page-anchor][/page-anchor]`. The string $PAGE$ will be replaced with the page number in the latter. The same example:

```
[more-list]
[first-anchor] First [/first-anchor]
[next-anchor] Forward | [/next-anchor]
[prev-anchor] Back [/prev-anchor]
[last-anchor] Last [/last-anchor]
[page-anchor] Page $PAGE$ (matches $MINPAGE$-$MAXPAGE$) | [/page-anchor]
[more]
[/more-list]
```

will display `Forward | Page 1 (matches 1-50) | Page 2 (matches 51-77) | Back` for 2 pages. Note that the following anchors are replaced with the page number, the minimum match on the page, and the maximum match on the page:

```
$PAGE$       Page number
$MINPAGE$    Minimum match on page
$MAXPAGE$    Maximum match on page
```

You can customize the HTML hyperlink with [link-template] [/link-template]. This is useful for adding a JavaScript onclick attribute, or setting the link target to a different window, etc.

```
[link-template]<a href="$URL$" target="_top">$ANCHOR$</a>[/link-template]
```

There are two tokens you can use as many times as needed in [link-template], which will be replaced as follows:

```
$URL$      The URL for the 'more' page in question
$ANCHOR$   The page number or the word "Next" or "Previous"
               for the link in question.
```

If have many pages of matches and don't wish to have all displayed at once, set `[decade-next][/decade-next]` and `[decade-prev][/decade-prev]`. If set them empty, a search with 31 pages will display pages 21-30 like:

```
Previous 1 2 3 4 5 6 7 8 9 10 [more>>] Next
```

and pages 11-20 like:

```
Previous [<<more] 11 12 13 14 15 16 17 18 19 20 [more>>] Next
```

If set to `[decade-next](higher)[/decade-next]` and `[decade-prev](lower)[/decade-prev]`, the following will be displayed:

```
Previous (lower) 11 12 13 14 15 16 17 18 19 20 (higher) Next
```

Of course, image−based anchors can be used as well.

[/more−list]

Companion to `[more−list]`.

[more]

Inserts a series of hyperlinks that will call up the next matches in a series. They look like this:

```
Previous 1 2 3 4 5 6 Next
```

The current page will not be a hyperlink. Every time the new link is pressed, the list is re−built to correspond to the current page. If there is no `Next` or `Previous` page, that link will not be shown.
See the `search.html` file for examples. Make sure to insert this item between a `[more−list]` and `[/more−list]` element pair.

[process−search]

Outputs the complete URL for a search, including Interchange session tags. Used as the ACTION value for the search form. This is exactly the same as [area search].

6. Sorting

Interchange has standard sorting options for sorting the search lists, loop lists, and item lists based on the contents of database fields. In addition, it adds list slices for limiting the displayed entries based on a start value and chunk size (or start and end value, from which a chunk size is determined). All accept a standard format sort tag which must be directly after the list call:

```
[loop 4 3 2 1]
[sort -2 +2]
    [loop-code]
[/loop]

[search-list]
[sort products:category:f]
    [item-price] [item-description]<BR>
[/search-list]

[item-list]
[sort products:price:rn]
    [item-price] [item-code]<BR>
[/item-list]

[loop search="ra=yes"]
[sort products:category products:title]
[loop-field category] [loop-field title] <BR>
[/loop]
```

All sort situations, `[search list]`, `[loop list]`, `[tag each table]`, and `[item-list]`, take options of the form:

```
[sort database:field:option* -n +n =n-n ... ]
```

database

The Interchange database identifier. This must be supplied and should normally be 'products' if using the default name for the database.

field

The field (column) of the database to be sorted on.

option

None, any, or combinations of the options:

```
f   case-insensitive sort (folded) (mutually exclusive of n)
n   numeric order (mutually exclusive of f)
r   reverse sort
```

-n

The starting point of the list to be displayed, beginning at 1 for the first entry.

+n

The number of entries to display in this list segment.

=n−n

The starting and ending point of the list display. This is an alternative to −n and +n. They should be specified in only one form. If both are specified, the last one will take effect.

...

Don't really put `...` in. This means that many sort levels are specified. Lots of sort levels with large databases will be quite slow.

Multiple levels of sort are supported, and database boundaries on different sort levels can be crossed. Cross−database sorts on the same level are not supported. If using multiple product databases, they must be sorted with embedded Perl. This is actually a feature in some cases, all items in a `used` database can be displayed before or after new ones in `products`.

Examples, all based on the `simple` demo:

Loop list

```
[loop 00-0011 19-202 34-101 99-102]
[sort products:title]
    [loop-code] [loop-field title]<BR>
[/loop]
```

Will display:

```
34-101 Family Portrait
00-0011 Mona Lisa
19-202 Radioactive Cats
99-102 The Art Store T-Shirt
```

\Alternatively:

```
[loop 00-0011 19-202 34-101 99-102]
[sort products:title -3 +2]
    [loop-code] [loop-field title]<BR>
[/loop]
```

\Displays:

```
19-202 Radioactive Cats
99-102 The Art Store T-Shirt
```

The tag `[sort products:title =3−4]` is equivalent to the above.

Search list

A search of all products (i.e., http://yoursystem.com/cgi−bin/simple/scan/ra=yes):

```
[search-list]
[sort products:artist products:title:rf]
    [item-field artist] [item-field title]<BR>
```

```
[/search-list]
```

will display:

```
Gilded Frame
Grant Wood American Gothic
Jean Langan Family Portrait
Leonardo Da Vinci Mona Lisa
Salvador Dali Persistence of Memory
Sandy Skoglund Radioactive Cats
The Art Store The Art Store T-Shirt
Vincent Van Gogh The Starry Night
Vincent Van Gogh Sunflowers
```

Note the reversed order of the title for Van Gogh and the presence of the accessory item Gilded Frame at the front of the list. It has no artist field and, as such, sorts first).
Adding a slice option:

```
[search-list]
[sort products:artist products:title:rf =6-10]
    [item-field artist] [item-field title]<BR>
[/search-list]
```

will display:

```
Sandy Skoglund Radioactive Cats
The Art Store The Art Store T-Shirt
Vincent Van Gogh The Starry Night
Vincent Van Gogh Sunflowers
```

If the end value/chunk size exceeds the size of the list, only the elements that exist will be displayed, starting from the start value.

Shopping cart

```
[item-list]
[sort products:price:rn]
    [item-price] [item-code]<BR>
[/item-list]
```

will display the items in the shopping cart sorted on their price, with the most expensive shown first. NOTE: This is based on the database field and doesn't take quantity price breaks or discounts into effect. Modifier values or quantities cannot be sorted.

Complete database contents

```
[tag each products]
[sort products:category products:title]
[loop-field category] [loop-field title] <BR>
[/tag]
```

A two level sort that will sort products based first on their category, then on their title within the category.

Note that large lists may take some time to sort. If a product database contains many thousands of items, using the `[tag each products]` sort is not recommended unless planning on caching or statically building

pages.

7. Shipping

Interchange has a powerful custom shipping facility that performs UPS and other shipper lookups, as well as a flexible rule−based facility for figuring cost by other methods.

7.1. Shipping Cost Database

The shipping cost database (located in ProductDir/shipping.asc) is a tab−separated ASCII file with eight fields: code, text description, criteria (quantity or weight, for example), minimum number, maximum number, and cost, query, and options. None of the fields are case−sensitive.

To define the shipping database in a catalog configuration file, set the Variable `MV_SHIPPING` to what would be its contents.

To set the file to be something other than `shipping.asc` in the products directory, set the `Special` directive:

```
Special  shipping.asc  /home/user/somewhere/shipping_defs
```

There are two styles of setting which can be mixed in the same file. The first is line−based and expects six or more TAB−separated fields. They would look like:

```
default No shipping weight  0   99999999    0

upsg    UPS Ground  weight  0   0   e Nothing to ship!
upsg    UPS Ground  weight  0   150 u Ground [default zip 98366] 3.00
upsg    UPS Ground  weight  150 999999  e @@TOTAL@@ lbs too heavy for UPS
```

The second is a freeform method with a `mode: Description text` introducing the mode line. The special encoding is called out by indented parameters. The below is identical to the above:

```
upsg: UPS Ground
    criteria    weight
    min         0
    max         0
    cost        e Nothing to ship!

    min         0
    max         150
    cost        u
    table       2ndDayAir
    geo         zip
    default_geo 98366
    adder       3

    min         150
    max         999999
    cost        e @@TOTAL@@ lbs too heavy for UPS
```

The second format has several advantages. Multiple lines can be spanned with the <<HERE document format, like so:

```
upsg: UPS Ground
    criteria    <<EOF
```

```
[perl]
    return 'weight' if $Values->{country} eq 'US';
    return 'weight' if ! $Values->{country};
    # Return blank, don't want UPS
    return '';
[/perl]
EOF
```

The definable fields are, in order, for the tab-separated format:

MODE

The unique identifier for that shipping method. It may be repeated as many times as needed.

DESCRIPTION

Text to describe the method (can be accessed on a page with the `[shipping-description]` element).

CRITERIA

Whether shipping is based on weight, quantity, price, etc. Valid Interchange tags can be placed in the field to do a dynamic lookup. If a number is returned, that is used as the accumulated criteria. That is, the total of weight, quantity, or price as applied to all items in the shopping cart.
See `Criteria Determination` below.

MINIMUM

The low bound of quantity/weight/criteria this entry applies to.

MAXIMUM

The high bound of quantity/weight/criteria this entry applies to. The first found entry is used in case of ties.

COST

The method of developing cost. It can be a number which will be used directly as the shipping cost, or a function, determined by a single character at the beginning of the field:

```
f       Formula (ITL tags OK, evaluated as Perl)
x       Multiplied by a number
[uA-Z]  UPS-style lookup
m       Interchange chained cost lookup (all items summed together)
i       Interchange chained cost lookup (items summed individually)
```

NEXT

The `next` field supplies an alternative shipping mode to substitute if the cost of the current one is zero.

ZONE

The UPS zone that is being defined.

QUERY

Interchange tags which will return a SQL query to select lines matching this specification. The current mode is replaced with this selection. If there is a query parameter of ?, it will be replaced with the mode name.

QUAL

The geographic qualification (if any) for this mode.

PERL

Perl code that is read and determines the criterion, not the cost. Use the `cost` option with "f" as the prelim to supply Perl code to determine cost.

TOTAL

Set to the accumulated criterion before passing to Perl.

OPT

Used to maintain UPS and freeform options. Normally these are set by separate lines in the shipping definition.

7.2. Criteria Determination

The criteria field varies according to whether it is the first field in the shipping file exactly matching the mode identifier. In that case, it is called the main criterion. If it is in subsidiary shipping lines matching the mode (with optional appended digits), it is called a qualifying criterion. The difference is that the main criterion returns the basis for the calculation (i.e., weight or quantity), while the qualifying criterion determines whether the individual line may match the conditions.

The return must be one of:

quantity

The literal value quantity as the main criterion will simply count the number of items in the shopping cart and return it as the accumulated criteria. If using a database table field named `quantity`, use the `table::field` notation.

o <field name> or <table>::<field name>

A valid database field (column) name as main criterion will cause the number of items in the shopping cart to be multiplied by the value of the field for each item to obtain the accumulated criteria. If the table is not supplied, defaults to the first `ProductFiles` table.

o n.nn

Where **n.nn** is any number, it will be directly used as the accumulated criteria. This can be effectively returned from a Perl subroutine or Interchange `[calc][item-list] ... [/item-list][/calc]` to create custom shipping routines.

IMPORTANT NOTE: The above only applies to the first field that matches the shipping mode exactly. Following criteria fields contain qualifier matching strings.

7.3. Shipping Calculation Modes

There are eight ways that shipping cost may be calculated. The method used depends on the first character of the `cost` field in the shipping database.

N.NN (digits)

If the first character is a digit, a number is assumed and read directly as the shipping cost.

e

If the first character is an `e`, a cost of zero is returned and an error message is placed in the session value `ship_message` (i.e., `[data session ship_message]` or $Session->{ship_message}).

f

If the character `f` is the first, Interchange will first interpret the text for any Interchange tags and then interpret the result as a formula. It is read as Perl code; the entire set of Interchange objects may be referenced with the code.

i

Specifies a chained shipping lookup which will be applied to each item in the shopping cart.

m

Specifies a chained shipping lookup which will be applied to the entire shopping cart.

u

Calls the UPS-style lookup. Can pre-define as many as desired. Though if want to do the hundreds available, it is best done on-the-fly.

x

If an `x` is first, a number is expected and is applied as a fixed multiplier for the accumulated criterion (@@TOTAL@@).

A-Z

If the first character is a capital letter, calls one of the 26 secondary UPS-style lookup zones. (Deprecated now that zones can be named directly).

7.4. How Shipping is Calculated

1. The base code is selected by reading the value of `mv_shipmode` in the user session. If it has not

been explicitly set, either by means of the DefaultShipping directive or by setting the variable on a form (or in an order profile), it will be `default`.
The mv_shipmode must be in the character class [A−Za−z0−9_]. If there are spaces, commas, or nulls in the value, they will be read as multiple shipping modes.
The criterion field is found. If it is quantity, it is the total quantity of items on the order form. If it is any other name, the criterion is calculated by multiplying the return value from the product database field for each item in the shopping cart, multiplied by its quantity. If the lookup fails due to the column or row not existing, a zero cost will be returned and an error is sent to the catalog error log. If a number is returned from an Interchange tag, that number is used directly.
Entries in the shipping database that begin with the same string as the shipping mode are examined. If none is found, a zero cost is returned and an error is sent to the catalog error log.

Note: The same mode name may be used for all lines in the same group, but the first one will contain the main criteria.

---

1. The value of the accumulated criteria is examined. If it falls within the minimum and maximum, the cost is applied.
2. If the cost is fixed, it is simply added.
3. If the cost field begins with an `x`, the cost is multiplied by the accumulated criterion, i.e., price, weight, etc.
4. If the cost field begins with `f`, the formula following is applied. Use @@TOTAL@@ as the value of the accumulated criterion.
5. If the cost field begins with `u` or a single letter from A−Z, a UPS−style lookup is done.
6. If the cost field begins with `s`, a Perl subroutine call is made.
7. If the cost field begins with `e`, zero cost is returned and an error placed in the session **ship_message** field, available as `[data session ship_message]`.

Here is an example shipping file using all of the methods of determining shipping cost.

Note: The columns are lined up for reading convenience. The actual entries should have **one** tab between fields.

```
global Option   n/a               0   0  g PriceDivide

rpsg  RPS     quantity         0   0     R RPS products/rps.csv
rpsg  RPS     quantity         0   5     7.00
rpsg  RPS     quantity         6   10    10.00
rpsg  RPS     quantity         11  150   x .95

usps  US Post price            0   0     0
usps  US Post price            0   50    f 7 + (1 * @@TOTAL@@ / 10)
usps  US Post price            50  100   f 12 + (.90 * @@TOTAL@@ / 10)
usps  US Post price            100 99999 f @@TOTAL@@ * .05

upsg  UPS     weight [value state]  0   0   e Nothing to ship.
upsg  UPS     AK HI            0   150   u upsg [default zip 980] 12.00 round
upsg  UPS                      0   150   u Ground [default zip 980] 2.00 round
upsg  UPS                      150 9999  e @@TOTAL@@ lb too heavy for UPS

upsca UPS/CA  weight           0   0     c C UPS_Canada products/can.csv
upsca UPS/CA  weight           −1   −1   o PriceDivide=0
upsca UPS/CA  weight           0   150   C upsca [default zip A7G] 5.00
upsca UPS/CA  weight           150 99999 e @@TOTAL@@ lb too heavy for UPS
```

global

This is a global option setting, called out by the `g` at the beginning. PriceDivide tells the shipping routines to multiply all shipping settings by the PriceDivide factor, except those explicitly set differently with the `o` individual modifier. This allows currency conversion. (Currently the only option is PriceDivide.)

rpsg

If the user selected RPS, (code rpsg) and the quantity on the order was 3, the cost of 7.00 from the second rpsg line would be applied. If the quantity were 7, the next entry from the third rpsg line would be selected for a cost of 10.00. If the quantity were 15, the last rpsg would be selected and the quantity of 15 multiplied by 0.95, for a total cost of 14.25.

usps

The next mode, `usps`, is a more complicated formula using price as the criteria. If the total price of all items in the shopping cart (same as `[subtotal]` without quantity price breaks in place) is from 1 to 50, the cost will be 7.00 plus 10 percent of the order. If the total is from 50.01 to 100, the cost will be 12.00 plus 9 percent of the order total. If the cost is 100.01 or greater, 5 percent of the order total will be used as the shipping cost.

upsg

The next, `upsg`, is a special case. It specifies a UPS lookup based on the store's UPS zone and two required values (and two optional arguments):

```
1. Weight
2. The zip/postal code of the recipient of which only
   the first three digits are used.
3. A fixed amount to add to the cost found in the UPS
   tables (use 0 as a placeholder if specifying roundup)
4. If set to 'round,' will round the cost up to the next
   integer monetary unit.
```

If the cost returned is zero, the reason will be placed as an error message in the session variable ship_message (available as `[data session ship_message]`).
UPS weights are always rounded up if any fraction is present.
The routines use standard UPS lookup tables. First, the UPS Zone file must be present. That is a standard UPS document specific to the retailer's area that must be obtained from UPS. It is entered into and made available to Interchange in TAB−delimited format. (As of March 1997, use the standard .csv file distributed by UPS on their Web site at `www.ups.com`.) Specify it with the UpsZoneFile directive. It is usually named something like `NNN.csv`, where NNN is the first three digits of the originating zip code. If placed in the products directory, the directive would look like:

```
UPSZoneFile  products/450.csv
```

Second, obtain the cost tables from UPS (again, get them from `www.ups.com`) and place them into an Interchange database. That database, its identifier specified with the first argument (Ground in the example) of the cost specification, is consulted to determine the UPS cost for that weight and rate schedule.
In the example below, use a database specification like:

```
Database  Ground  Ground.csv  CSV
```

A simple shipping cost qualification can be appended to a UPS lookup. If any additional parameters are present after the five usual ones used for UPS lookup, they will be interpreted as a Perl subroutine call. The syntax is the same as if it was encased in the tag [perl] [/perl], but the following substitutions are made prior to the call:

```
@@COST@@  is replaced with whatever the UPS lookup returned
@@GEO@@   is replaced with the zip (or other geo code)
@@ADDER@@ is replaced with the defined adder
@@TYPE@@  is replaced with the UPS shipping type
@@TOTAL@@ is replaced with the total weight
```

The example above also illustrates geographic qualification. If the value of the form variable state on the checkout form is AK or HI, the U.S. states Alaska and Hawaii, a $10.00 additional charge (over and above the normal $2.00 handling charge) is made. This can also be used to select on country, product type, or any other qualification that can be encoded in the file.

upsca

The next entry is just like the UPS definition except it defines a different lookup zone file (`products/can.csv`) and uses a different database, `upsca`. It also disables the global PriceDivide option for itself only, not allowing currency conversion. Otherwise, the process is the same.

Up to 27 different lookup zones can be defined in the same fashion, allowing for multiple zone files. If one of the cost lines (the last field) in the `shipping.asc` file begins with a `c`, it configures another lookup zone which must be lettered from `A to Z`. It takes the format:

```
c X name file* length* multiplier*
```

where X is the letter from `A–Z`. The name is used internally as an identifier and must be present. The optional `file` is relative to the catalog root (like `UpsZoneFile` is). If it is not present, the file equal to `name` in the products directory (ProductDir) will be used as the zone file. If the optional digit `length` is present, that determines the number of significant digits in the passed postal/geo code.
When the optional `multiplier` is present, the weight is multiplied by it before doing the table lookup. This allows shipping weights in pounds or kilograms to be adapted to a table using the opposite as the key. Remember, the match on weight must be exact, and Interchange rounds the weight up to the next even unit. To define the exact equivalent of the UPS lookup zone, do the following:

```
c U UPS products/450.csv 3 1
```

The only difference is that the beginning code to call the lookup is upper–case `U` instead of lower–case `u`.

7.5. More On UPS–Style Lookup

The UPS–style lookup uses two files for its purposes, both of which need to be in a format like UPS distributes for US shippers.

The zone file is a file that is usually specific to the originating location. For US shippers shipping to US locations, it is named for the first three digits of the originating zip code with a CSV extension. For example, `450.csv`.

It has a format similar to:

```
low − high, zone,zone,zone,zone
```

The `low` entry is the low bound of the geographic location; `high` is the high bound. (By geographic location, the zip code is meant.) If the first digits of the zip code, compared alphanumerically, fall between the low and high values, that zone is used as the column name for a lookup in the rate database. The weight is used as the row key.

The first operative row of the zone file (one without leading quotes) is used to determine the zone column name. In the US, it looks something like:

```
Dest. ZIP,Ground,3 Day Select,2nd Day Air,2nd Day Air A.M.,\
 Next Day Air Saver,Next Day Air
```

Interchange strips all non−alpha characters and comes up with:

```
DestZIP,Ground,3DaySelect,2ndDayAir,2ndDayAirAM,NextDayAirSaver,NextDayAir
```

Therefore, the zone column (shipping type) that would be used for UPS ground would be "Ground," and that is what the database should be named. To support the above, use a `shipping.asc` line that reads:

```
upsg  UPS Ground  weight  0  150  u Ground [default zip 983]
```

and a `catalog.cfg` database callout of:

```
Database  Ground  Ground.csv  CSV
```

These column names can be changed as long as they correspond to the `identifier` of the rate database.

The rate database is a standard Interchange database. For U.S. shippers, UPS distributes their rates in a fairly standard comma−separated value format, with weight being the first (or key) column and the remainder of the columns corresponding to the zone which was obtained from the lookup in the zone file.

To adapt other shipper zone files to Interchange's lookup, they will need to fit the UPS US format. (Most of the UPS international files don't follow the U.S. format). For example, the 1998 Ohio−US to Canada file begins:

```
Canada Standard Zone Charts from Ohio
Locate the zone by cross−referencing the first three
characters of the destination Postal Code in the Postal
Range column.

Postal Range  Zone
A0A  A9Z      54
B0A  B9Z      54
C0A  C9Z      54
E0A  E9Z      54
G0A  G0A      51
G0B  G0L      54
G0M  G0S      51
G0T  G0W      54
```

It will need to be changed to:

```
Destination,canstnd
A0A−A9Z, 54
```

```
B0A−B9Z, 54
C0A−C9Z, 54
E0A−E9Z, 54
G0A−G0A, 51
G0B−G0L, 54
G0M−G0S, 51
G0T−G0W, 54
```

Match it with a `canstnd` CSV database that looks like this:

```
Weight,51,52,53,54,55,56
1,7.00,7.05,7.10,11.40,11.45,11.50
2,7.55,7.65,7.75,11.95,12.05,12.10
3,8.10,8.15,8.40,12.60,12.70,12.85
4,8.65,8.70,9.00,13.20,13.30,13.55
5,9.20,9.25,9.75,13.85,13.85,14.20
6,9.70,9.85,10.35,14.45,14.50,14.90
7,10.25,10.40,11.10,15.15,15.15,15.70
8,10.80,10.95,11.70,15.70,15.75,16.35
9,11.35,11.55,12.30,16.40,16.45,17.20
```

It is called out in catalog.cfg with:

```
Database canstnd canstnd.csv CSV
```

With the above, a 4−pound shipment to postal code E5C 4TL would yield a cost of 13.20.

7.6. Geographic Qualification

If the return value in the main criterion includes whitespace, the remaining information in the field is used as a qualifier for the subsidiary shipping modes. This can be used to create geographic qualifications for shipping, as in:

```
upsg  UPS Ground  weight [value state]  0   0    e No items selected
upsg  UPS Ground  AK HI                 0   150  u Ground [value zip] 12.00
upsg  UPS Ground                       0   150  u Ground [value zip] 3.00
```

If `upsg` is the mode selected, the value of the user session variable `state` is examined to see if it matches the geographic qualification on a whole−word boundary. If it is `AK` or `HI`, UPS Ground with an adder of 12 will be selected. If it "falls through," UPS Ground with an adder of 3 will be selected.

7.7. Handling Charges

Additional handling charges can be defined in the shipping file by setting the form variable `mv_handling` to a space, comma, or null−separated set of valid shipping modes. The lookup and charges are created in the same fashion, and the additional charges are added to the order. (The user is responsible for displaying the charge on the order report or receipt with a `[shipping handling]` tag, or the like.) All of the shipping modes found in mv_handling will be applied. If multiple instances are found on a form, the accordingly null−separated values will all be applied. NOTE: This should not be done in an item−list unless the multiple setting of the variables is accounted for.

To only process a handling charge once, do the following:

```
[item−list]
```

```
[if-item-field very_heavy]
[perl values]
    return '' if $Values->{mv_handling} =~ /very_heavy/;
    return "<INPUT TYPE=hidden NAME=mv_handling VALUE=very_heavy>";
[/perl]
[/if-item-field]
[/item-list]
```

A non-blank/non-zero value in the database field will trigger Perl code which will only set `mv_handling` once.

7.8. Default Shipping Mode

If a default shipping mode other than `default` is desired, enter it into the `DefaultShipping` directive:

```
DefaultShipping     upsg
```

This will make the entry on the order form checked by default when the user starts the order process, if it is put in the form:

```
<INPUT TYPE=RADIO NAME=mv_shipmode VALUE=upsg [checked mv_shipmode upsg]>
```

To force a choice by the user, make `mv_shipmode` a required form variable (with RequiredFields or in an order profile) and set `DefaultShipping` to zero.

8. User Database

Interchange has a user database function which allows customers to save any pertinent values from their session. It also allows the setting of database or file access control lists for use in controlling access to pages and databases on a user−by−user basis.

The database field names in the user database correspond with the form variable names in the user session. If there is a column named `address`, when the user logs in the contents of that field will be placed in the form variable `address`, and will be available for display with `[value address]`. Similarly, the database value is available with `[data table=userdb column=address key=username]`.

The ASCII file for the database will not reflect changes unless the file is exported with `[tag export userdb][/tag]`. It is not advisable to edit the ASCII file, as it will overwrite the real data that is in the DBM table. User logins and changes would be lost. Note: This would not happen with SQL, but editing the ASCII file would have no effect. It is recommended that the NoImport configuration directive be set accordingly.

The field names to be used are not set in concrete. They may be changed with options. Fields may be added or subtracted at any time. Most users will choose to keep the default demo fields for simplicity sake, as they cover most common needs. As distributed in the demo, the fields are:

```
code
accounts
acl
address
address_book
b_address
b_city
b_country
b_name
b_nickname
b_phone
b_state
b_zip
carts
city
country
db_acl
email
email_copy
fax
fax_order
file_acl
mv_credit_card_exp_month
mv_credit_card_exp_year
mv_credit_card_info
mv_credit_card_type
mv_shipmode
name
order_numbers
p_nickname
password
phone_day
phone_night
preferences
s_nickname
```

```
state
time
zip
```

A few of those fields are special in naming, though all can be changed via an option. A couple of the fields are reserved for Interchange's use.

Note: If not running with PGP or other encryption for credit card numbers, which is never recommended, it is important that the mv_credit_card_info field be removed from the database.

The special database fields are:

```
accounts         Storage for billing accounts book
address_book     Storage for shipping address book
b_nickname       Nickname of current billing account
carts            Storage for shopping carts
p_nickname       Nickname for current preferences
preferences      Storage for preferences
s_nickname       Nickname for current shipping address
db_acl           Storage for database access control lists
file_acl         Storage for file access control lists
acl              Storage for simple integrated access control
```

If not defined, the corresponding capability is not available.

Note: The fields `accounts`, `address_book`, `carts`, and `preferences` should be defined as a BLOB type, if using SQL. This is also suggested for the acl fields if those lists could be large.

Reserved fields include:

```
code        The username (key for the database)
password    Password storage
time        Last time of login
```

8.1. The [userdb ...] Tag

Interchange provides a `[userdb ...]` tag to access the UserDB functions.

```
[userdb
       function=function_name
       username="username"*
       assign_username=1
       username_mask=REGEX*
       password="password"*
       verify="password"*
       oldpass="old password"*
       crypt="1|0"*
       shipping="fields for shipping save"
       billing="fields for billing save"
       preferences="fields for preferences save"
       ignore_case="1|0"*
       force_lower=1
       param1=value*
       param2=value*
       ...
       ]
```

* Optional

It is normally called in an `mv_click` or `mv_check` setting, as in:

```
[set Login]
mv_todo=return
mv_nextpage=welcome
[userdb function=login]
[/set]

<FORM ACTION="[process]" METHOD=POST>
<INPUT TYPE=hidden NAME=mv_click VALUE=Login>
Username <INPUT NAME=mv_username SIZE=10>
Password <INPUT NAME=mv_password SIZE=10>
</FORM>
```

There are several global parameters that apply to any use of the `userdb` functions. Most importantly, by default, the database table is set to be userdb. If another table name must be used, include a `database=table` parameter with any call to `userdb`. The global parameters (default in parentheses):

```
database          Sets user database table (userdb)
show              Show the return value of certain functions
                  or the error message, if any (0)
force_lower       Force possibly upper-case database fields
                  to lower case session variable names (0)
billing           Set the billing fields (see Accounts)
shipping          Set the shipping fields (see Address Book)
preferences       Set the preferences fields (see Preferences)
bill_field        Set field name for accounts (accounts)
addr_field        Set field name for address book (address_book)
pref_field        Set field name for preferences (preferences)
cart_field        Set field name for cart storage (carts)
pass_field        Set field name for password (password)
time_field        Set field for storing last login time (time)
outboard          Set fields that live in another table
outboard_key_col  Set field providing key for outboard tables
expire_field      Set field for expiration date (expire_date)
acl               Set field for simple access control storage (acl)
file_acl          Set field for file access control storage (file_acl)
db_acl            Set field for database access control storage (db_acl)
indirect_login    Log in field if different than real username ('')
```

By default the system crypt() call will be used to compare the password. This is best for security, but the passwords in the user database will not be human readable.

If no critical information is kept and Interchange administration is not done via the `UserDB` capability, use the `UserDB` directive (described below) to set encryption off by default:

```
UserDB   default   crypt   0
```

Encryption can still be set on by passing `crypt=1` with any call to a `new_account`, `change_pass`, or `login` call.

If you are encrypting, and you wish to use MD5 to encrypt the passwords, set the `md5` parameter:

UserDB default md5 1

8.2. Setting Defaults with the UserDB Directive

The `UserDB` directive provides a way to set defaults for the user database. For example, to save and recall the scratch variable `tickets` in the user database instead of the form variable `tickets`, set:

```
UserDB   default   scratch  tickets
```

That makes every call to `[userdb function=login]` equivalent to `[userdb function=login scratch=tickets]`.

To override that default for one call only, use `[userdb function=login scratch="passes"]`.

To log failed access authorizations, set the `UserDB` profile parameter `log_failed` true:

```
UserDB  default  log_failed 1
```

To disable logging of failed access authorizations (the default), set the `UserDB` profile parameter `log_failed` to 0:

```
UserDB  default  log_failed 0
```

The `UserDB` directive uses the same key−value pair settings as the `Locale` and `Route` directives. If there are more than one set of defaults, set them in a hash structure:

```
UserDB  crypt_case  <<EOF
{
    'scratch'        => 'tickets',
    'crypt'          => '1',
    'ignore_case'    => '0',
}
EOF

UserDB  default  <<EOF
{
    'scratch'        => 'tickets',
    'crypt'          => '1',
    'ignore_case'    => '1',
}
EOF
```

Note: The usual here−document caveats apply. The "EOF" must be on a line by itself with no leading/trailing whitespace.

The last one to be set becomes the default.

The option `profile` selects the set to use. For usernames and passwords to be case sensitive with no encryption, pass this call:

```
[userdb function=new_account profile=case_crypt]
```

The username and password will be stored as typed in, and the password will be encrypted in the database.

8.3. User Database Functions

The user database features are implemented as a series of functions attached to the `userdb` tag. The functions are:

login

Active parameters: username, password, crypt, md5, pass_field, ignore_case, indirect_login
Log in to Interchange. By default, the username is contained in the form variable `mv_username` and the password in `mv_password`. If the login is successful, the session value `username` (`[data session username]`) will be set to the user name. If `indirect_login` is used, it should be set to a field name which can be used as a lookup for the real username. This also causes a `new_account` operation to create a user account based on an assigned username, and `assign_username` should always be set when using indirect login.
This will recall the values of all non−special fields in the user database and place them in their corresponding user form variables.
The `CookieLogin` directive (catalog.cfg) allows users to save their username/password in a cookie. Expiration time is set by `SaveExpire`, renewed every time they log in. To cause the cookie to be generated originally, the form variable `mv_cookie_password` or `mv_cookie_username` must be set in the login form. The former causes both username and password to be saved, the latter just the username.

logout

Log out of Interchange. No additional parameters are needed.

new_account

Active parameters: username, password, verify, assign_username, username_mask, ignore_case,indirect_login
Create a new account. It requires the `username`, `password`, and `verify` parameters, which are by default contained in the form variables `mv_username`, `mv_password`, `mv_verify` respectively.
If the `assign_username` parameter is set, `UserDB` will assign a sequential username. The `counter` parameter can be used to set the filename (must be absolute), or the default of CATALOG_DIR/etc/username.counter can be accepted. The first username will be "U0001" if the counter doesn't exist already.
If `assign_username` is used, you can choose to have a pseudo−username that is different from the real username. (Email address is commonly used.) The field name is contained in the `indirect_login` parameter. When the user logs in this field name will also be used to find the real username. The value must be unique in the database or a "user already exists" error will be thrown.
The `ignore_case` parameter forces the username and password to lower case in the database, in effect rendering the username and password case−insensitive. This is recommended if using email address as a login.
If `username_mask` is set to a valid Perl regular expression (without the surrounding / /), then any username containing a matching string will not be allowed for use. For example, to screen out order numbers from being used by a random user:

```
[userdb function=new_account
        username_mask="^[A-Z]*[0-9]"
        ]
```

The `CookieLogin` directive (catalog.cfg) allows users to save their username/password in a cookie. Expiration time is set by `SaveExpire`, renewed every time they log in. To cause the cookie to be generated

originally, the form variable `mv_cookie_password` or `mv_cookie_username` must be set in the login form. The former causes both username and password to be saved, the latter just the username.
To automatically create an account for every order, set the following in the `OrderReport` file:

```
[userdb function=new_account
        username="[value mv_order_number]"
        password="[value zip]"
        verify="[value zip]"
        database="orders"
        ]
```

This would be coupled with a login form that asks for order number and zip code, thereupon allowing the display of the contents of a transaction database with (presumably updated) order status information or a shipping company tracking number.

change_pass

Active parameters: username, password, verify, oldpass
Change the password on the currently logged−in account. It requires the `username`, `password`, `verify`, and `oldpass` parameters, which are by default contained in the form variables `mv_username`, `mv_password`, `mv_verify`, `mv_password_old` respectively.

set_shipping

Active parameters: nickname, shipping, ship_field
Place an entry in the shipping Address book. For example:

```
[userdb function=set_shipping nickname=Dad]
```

See Address Book below.

get_shipping

Active parameters: nickname, shipping, ship_field
Recall an entry from the shipping Address book. For example:

```
[userdb function=get_shipping nickname=Dad]
```

See Address Book below.

get_shipping_names

Active parameters: ship_field
Gets the names of shipping address book entries and places them in the variable `address_book`. By default, it does not return the values. To have the values returned, set the parameter `show` to 1, as in:

```
[set name=shipping_nicknames
     interpolate=1]
  [userdb function=get_shipping_names show=1]
[/set]
```

set_billing

Active parameters: nickname, billing, bill_field
Place an entry in the billing accounts book. For example:

```
[userdb function=set_billing nickname=discover]
```

See Accounts Book below.

get_billing

Active parameters: nickname, billing, bill_field
Recall an entry from the billing accounts book. For example:

```
[userdb function=get_billing nickname=visa]
```

See Accounts Book below.

save

Saves all non−special form values that have columns in the user database. If a field is defined as `scratch`, it retrieves the field from the Scratch storage area; otherwise from Values. If the field is one of the `outboard` fields, it will save it in the outboard table with the value of `outboard_key_col` as the key.

set_cart

Save the contents of a shopping cart.

```
[userdb function=set_cart nickname=christmas]
```

See Carts below.

get_cart

Active parameters: nickname, carts_field, target
Recall a saved shopping cart.

```
[userdb function=get_cart nickname=mom_birthday]
```

Setting `target` saves to a different shopping cart than the default main cart. The `carts_field` controls the database field used for storage.

set_acl

Active parameters: location, acl_field, delete
Set a simple acl. For example:

```
[userdb function=set_acl location=cartcfg/editcart]
```

This allows the current user to access the page "cartcfg/editcart" if it is access−protected.
To delete access, do:

```
[userdb function=set_acl location=cartcfg/editcart delete=1]
```

To display the setting at the same time as setting, use the `show` attribute:

```
[userdb function=set_acl location=cartcf/editcart show=1]
```

check_acl

Active parameters: location, acl_field
Checks the simple access control listing for a location, returning 1 if allowed and the empty string if not allowed.

```
[if type=explicit
    compare="[userdb
                function=check_acl
                location=cartcfg/editcart]"
]
[page cartcfg/editcart]Edit your cart configuration</a>
[/if]
```

set_file_acl, set_db_acl

Active parameters: location, mode, db_acl_field, file_acl_field, delete
Sets a complex access control value. Takes the form:

```
[userdb function=set_file_acl
        mode=rw
        location=products/inventory.txt]
```

where mode is any value to be checked with `check_file_acl`. As with the simple ACL, use delete=1 to delete the location entirely.

check_file_acl, check_db_acl

Active parameters: location, mode, db_acl_field, file_acl_field
Checks a complex access control value and returns a true/false (1/0) value. Takes the form:

```
[userdb function=check_db_acl
        mode=w
        location=inventory]
```

where mode is any value to be checked with `check_file_acl`. It will return true, if the mode string is contained within the entry for that location. For example:

```
[if type=explicit
    compare="[userdb
                function=check_db_acl
                mode=w
                location=inventory]"
]
[userdb function=set_acl location=cartcfg/edit_inventory]
[page cartcfg/edit_inventory]You may edit the inventory database</a>
[else]
[userdb function=set_acl location=cartcfg/edit_inventory delete=1]
Sorry, you can't edit inventory.
[/if]
```

8.4. Address Book

Address_book is a shipping address book. The shipping address book saves information relevant to shipping the order. In its simplest form, this can be the only address book needed. By default these form values are included:

```
s_nickname
name
fname
lname
address
address1
address2
address3
city
state
zip
country
phone_day
mv_shipmode
```

The first field is always the name of the form variable that contains the key for the entry. The values are saved with the `[userdb function=set_shipping]` tag call, and are recalled with `[userdb function=get_shipping]`. A list of the keys available is kept in the form value `address_book`, suitable for iteration in an HTML select box or in a set of links.

To get the names of the addresses, use the `get_shipping_names` function:

```
[userdb function=get_shipping_names]
```

By default, they are placed in the variable `address_book`. Here is a little snippet that builds a select box:

```
<FORM ACTION="[process]" METHOD=POST>
[userdb function=get_shipping_names]
[if value address_book]
<SELECT NAME="s_nickname">
[loop arg="[value address_book]"] <OPTION> [loop-code] [/loop]
</SELECT>
<INPUT TYPE=submit NAME=mv_check VALUE="Recall Shipping">
</FORM>
```

The same principle works with accounts, carts, and preferences.

To restore a cart based on the above, put in an `mv_check` routine:

```
[set Recall Shipping]
mv_todo=return
mv_nextpage=ord/basket
[userdb function=get_shipping nickname="[value s_nickname]"]
[/set]
```

When the `mv_check` variable is encountered, the contents of the scratch variable `Recall Shipping` are processed and the shipping address information inserted into the user form values. This is destructive of any current values of those user session variables, of course.

To change the fields that are recalled or saved, use the `shipping` parameter:

```
[userdb function=get_shipping
        nickname=city_and_state
        shipping="city state"]
```

Only the values of the `city` and `state` variables will be replaced.

8.5. Accounts Book

The accounts book saves information relevant to billing the order. By default these form values are included:

```
b_nickname
b_name
b_fname
b_lname
b_address
b_address1
b_address2
b_address3
b_city
b_state
b_zip
b_country
b_phone
purchase_order
mv_credit_card_type
mv_credit_card_exp_month
mv_credit_card_exp_year
mv_credit_card_info
```

The values are saved with the `[userdb function=set_billing]` tag call, and are recalled with `[userdb function=get_billing]`. A list of the keys available is kept in the form value `accounts`, suitable for iteration in an HTML select box or in a set of links.

8.6. Preferences

Preferences are miscellaneous session information. They include, by default, the following fields:

```
email
fax
phone_night
fax_order
email_copy
```

The field `p_nickname` acts as a key to select the preference set. To change the values that are included with the `preferences` parameter:

```
[userdb function=set_preferences
        preferences="email_copy email fax_order fax"]
```

or in catalog.cfg:

```
UserDB default preferences "mail_list email fax_order music_genre"
```

8.7. Carts

The contents of shopping carts may be saved or recalled in much the same fashion. See the Simple demo application `ord/basket.html` page for an example.

8.8. Controlling Page Access With UserDB

Interchange can implement a simple access control scheme with the user database. Controlled pages must reside in a directory which has a file named `.access` that is zero bytes in length. (If it is more than 0 bytes, only the RemoteUser or MasterHost may access files in that directory.)

Set the following variables in `catalog.cfg`:

```
Variable   MV_USERDB_ACL_TABLE  userdb
Variable   MV_USERDB_ACL_COLUMN acl
```

The `MV_USERDB_ACL_TABLE` is the table which controls access, and likewise the `MV_USERDB_ACL_TABLE` names the column in that database which will be checked for authorization.

The database entry should contain the complete Interchange−style page name of the page to be allowed. It will not match substrings.

For example, if the user `flycat` followed this link:

```
<A HREF="[area cartcfg/master_edit]">Edit</A>
```

Access would be allowed if the contents of the userdb were:

```
code    acl
flycat  cartcfg/master_edit
```

and disallowed if it were:

```
code    acl
flycat  cartcfg/master_editor
```

Access can be enabled with:

```
[userdb function=set_acl location="cartcfg/master_edit"]
```

Access can be disallowed with:

```
[userdb function=set_acl
        delete=1
        location="cartcfg/master_edit"]
```

Of course, a pre−existing database with the ACL values will work as well. It need not be in the UserDB setup.

8.9. Using more than one table

You can save/retrieve userdb information from more than one table with the `outboard` specifier. It is a quoted key−value comma−separated series of field specifications. For instance, if the billing address is to be stored in a separate table named "billing", you would do:

```
UserDB  default outboard  <<EOF
    "b_fname=billing::first_name,
     b_lname=billing::last_name,
     b_address1=billing::address1,
     b_address2=billing::address2,
                b_etc=billing::etc"
EOF
```

When the user logs in, Interchange will access the `first_name` field in table `billing` to get the value of `b_fname`. When the values are saved, it will be saved there as well. If you wish to make the fields read−only, just set `UserDB default scratch "b_fname b_lname ..."` and the values will be retrieved/saved from there. To initialize the values for a form, you could do a function after the user logs in:

```
[calc]
       my @s_fields = grep /\S/, split /[\s,\0]+/, $Config->{UserDB}{scratch};
       for(@s_fields) {
               $Values->{$_} = $Scratch->{$_};
       }
       return;
[/calc]
```

If the fields in the outboard table use another key besides `username`, you can specify the column in the userdb that contains the key value:

```
UserDB  default  outboard_key_col   account_id
```

9. Tracking and Back−End Order Entry

Interchange allows the entry of orders into a system through one of several methods. Orders can be written to an ASCII file or formatted precisely for email−based systems. Or they can go directly into an SQL or DBM database. Finally, embedded Perl allows completely flexible order entry, including real−time credit card verification and settlement.

9.1. ASCII Backup Order Tracking

If `AsciiTrack` is set to a legal file name (based in VendRoot unless it has a leading "/"), a copy of the order is saved and sent in an email.

If the file name string begins with a pipe "|", a program will be run and the output "piped" to that program. This allows easy back−end entry of orders with an external program.

9.2. Database Tracking

Once the order report is processed, the order is complete. Therefore, it is the ideal place to put Interchange tags that make order entries in database tables.

A good model is to place a single record in a database summarizing the order and a series of lines that correspond to each line item in the order. This can be in the same database table. If the order number itself is the key for the summary, a line number can be appended to the order number to show each line of the order.

The following would summarize a sample order number S00001 for part number 00−0011 and 99−102:

```
code     order_number part_number  quantity   price    shipping  tax
S00001   S00001                    3          2010     12.72     100.50
S00001−1 S00001       00−0011      2          1000     UPS       yes
S00001−2 S00001       99−102       1          10       UPS       yes
```

Fields can be added where needed, perhaps with order status, shipping tracking number, address, customer number, or other information.

The above is accomplished with Interchange's `[import ....]` tag using the convenient `NOTES` format:

```
[set import_status]
[import table=orders type=LINE continue=NOTES]

code: [value mv_order_number]
order_number: [value mv_order_number]
quantity: [nitems]
price: [subtotal noformat=1]
shipping: [shipping noformat=1]
tax: [salestax noformat=1]

[/import]

[item−list]
[import table=orders type=LINE continue=NOTES]

code: [value mv_order_number]−[item−increment]
order_number: [value mv_order_number]
```

```
quantity: [item-quantity]
price: [item-price noformat=1]
shipping: [shipping-description]
tax: [if-item-field nontaxable]No[else]Yes[/else][/if]

[/import][/item-list]
```

9.3. Order Routing

Interchange can send order emails and perform custom credit card charges and/or logging for each item. The Route directive is used to control this behavior, along with the `mv_order_route` item attribute.

If no Route is in the catalog, Interchange uses a default "mail out the order and show a receipt" model.

Routes are established with the `Route` directive, which is similar to the `Locale` directive. Each route is like a locale, so that key-value pairs can be set. Here is an example setting:

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

Note: Values with whitespace in them must be quoted.

You can also set the route in a valid Perl hash reference string:

```
Route  mail <<EOR
{
     pgp_key         => '0x67798115',
     email           => 'orders@akopia.com',
     reply           => 'service@akopia.com',
     encrypt         => 1,
     encrypt_program => q{/usr/bin/gpg -e -a -r '%s' --batch},
     report          => 'etc/report_mail',
}
EOR
```

This route would be used whenever the *mail* route was called by one of the three possible methods:

route called from master route

Called via the `cascade` parameter from the master route. This is the way that most routes are called in Interchange's the Foundation manpage demo. These routes treat the order as a whole.

route set in item

An item in the shopping cart has `mail` as the value in the attribute `mv_order_route`. This method is item-specific to this item (or group of items in route `mail`).

route set in the form variable mv_order_route

By setting a value in the `mv_order_route` form variable, you can specify one or more routes to run. This is the deprecated method used in earlier Interchange 4.6.x and Minivend 4 routes. It will still work fine.

The last route that is defined is the `master` route, by convention named *main*. Besides setting the global behavior of the routing, it provides some defaults for other routes. For example, if `encrypt_program` is set there, then the same value will be the default for all routes. Most settings do not fall through.

The attributes that can be set are:

attach

Determines whether the order report should be attached to the main order report e−mail. This is useful if certain items must be printed separately from others, perhaps for FAX to a fulfillment house.
cascade

A list of routes which should be pushed on the stack of routes to run, *after all currently scheduled routes are done*. NOTE: cascades can cause endless loops, so only one setting is recommended, that being the main route.

commit

Perl code which should be performed on a route commit.

commit_tables

Tables that are to be pre−opened before running the Perl commit code.

counter

The location of a counter file which should be used instead of `OrderCounter` for this route. It will generate a different value for `mv_order_number` for the route. This is normally used to obtain unique order references for multi−vendor routing.

credit_card

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

dynamic_routes

If set in the the master manpage route, will cause the the RouteDatabase manpage to be checked for a route. If it exists, it will be read in and the database copy used instead of the static copy build at catalog configuration time. If set in a subsidiary route, that route will be ignored during catalog.cfg, and `dynamic_routes` must be active for it to be seen.

email

The email address(es) where the order should be sent. Set just like the `MailOrderTo` directive, which is also the default.

empty

This should be set if neither attach or email is set.

encrypt

Whether the entire order should be encrypted with the **encrypt_program**. If `credit_card` is set, the credit card will first be encrypted, then the entire order encrypted.

encrypt_program

The encryption program incantation which should be used. Set identically to the `EncryptProgram` directive, except that %s will be replaced with the `pgp_key`. Default is empty.

errors_to

Sets the `Errors-To:` e-mail header so that bounced orders will go to the proper address. Default is the same as `MailOrderTo`.

expandable

If set in the the master manpage route, route settings will be expanded for ITL tags. No effect if the route is not the master.

extended

Extended route settings that take the form of an Interchange option list; normally a Perl hash reference that will be read. These settings always overwrite any that currently exist, regardless of the order in which they are specified. For example:

```
Route  main   extended    { email => 'milton@akopia.com' }
Route  main   email       papabear@minivend.com
```

The ultimate setting of `email` will be `milton@akopia.com`.

increment

Whether the order number should be incremented as a result of this result. Default is not to increment, as the order number should usually be the same for different routes within the same customer order.

individual_track

A directory where individual order tracking files will be placed. The file name will correspond to the value of `mv_order_number`. This can be useful for batching orders via download.

individual_track_ext

The extension that will be added to the file name for `individual_track`. Must contain a period (`.`), if that is desired.

```
individual_track_ext     .pgp
```

individual_track_mode

A number representing the final permission mode for the `individual_track` file. Usually expressed in octal:

```
individual_track_mode    0444
```

master

If set, this route becomes the master route for `supplant`, `dynamic_routes`, `errors_to`, and `expandable`, and supplies the setting for `receipt` and the `attach` report. Switching `master` in midstream is unlikely to be successful −− it should certainly be the first route in a cascade.

payment_mode

If this is set, enables a payment mode for the route. (Payment modes are also set in the `Route` directive.)

pgp_cc_key

The PGP/GPG key selector that is used to determine which public key is used for encryption of credit cards only. With PGP 5 and 6, see appropriate values by using the command `pgpk -l`. For GPG, use `gpg --list-keys`. Defaults to the value of the pgp_key manpage.

pgp_key

The PGP key selector that is used to determine which public key is used for encryption. If `pgp_cc_key` is set, that key will be used for credit card encryption instead of `pgp_key`. With PGP 5 and 6, see appropriate values by using the command `pgpk -l`. For GPG, use `gpg --list-keys`. Defaults to the value of the pgp_key manpage.

profile

The custom order profile which should be performed to check the order *prior* to actually running the route. If it fails, the route will not be performed. See `OrderProfile` and `mv_order_profile`.

receipt

The receipt page that should be used for this routing. This only applies if supplant is set for the route, and that normally would only be in the default route.

report

The report page that should be used for this routing. If `attach` is defined, the contents of the report will be placed in a MIME attachment in the main order report.

reply

The `Reply-To` header that should be set. Default is the same as `email`.
If there are only word characters (A−Za−z0−9 and underscore), it describes an Interchange variable name where the address can be found.

rollback

Perl code which should be performed on a route rollback.

rollback_tables

Tables that are to be pre−opened before running the Perl rollback code.

supplant

Whether the master route should supplant the main order report. If set, the `AsciiTrack` operation will use this route and the normal Interchange order e−mail sequence will not be performed. This is normally set in the master route.

track

The name of a file which should be used for tracking. If the `supplant` attribute is set, the normal order tracking will be used as well.

track_mode

A number representing the final permission mode for the `track` file. Usually expressed in octal:

```
track_mode    0444
```

transactions

A list of tables to put in transactions mode at the beginning of the route. Used to ensure that orders get rolled back if another route fails.
The *first* route to open a table must have this parameter, otherwise transactions will not work. If any route fails (except ones marked error_ok) then a rollback will be done on these tables. If all routes succeed, a commit will be performed at the end of all order routes.

Individual item routing causes all items labeled with that route to be placed in a special sub−cart that will be used for the order report. This means that the `[item−list] LIST [/item−list]` will only contain those items, allowing operations to be performed on subsets of the complete order. The `[subtotal]`, `[salestax]`, `[shipping]`, `[handling]`, and `[total−cost]` tags are also affected.

Here is an example of an order routing:

```
Route  HARD      pgp_key          0x67798115
Route  HARD      email            hardgoods@akopia.com
Route  HARD      reply            service@akopia.com
Route  HARD      encrypt          1
Route  HARD      report           etc/report_mail

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

Route  mail      pgp_key          0x67798115
Route  mail      email            orders@akopia.com
Route  mail      reply            service@akopia.com
Route  mail      encrypt          1
Route  mail      report           etc/report_all
```

```
Route  user      error_ok         1
Route  user      email            email
Route  user      reply            service@akopia.com
Route  user      report           etc/user_copy

Route  log       empty            1
Route  log       report           etc/log_transaction
Route  log       transactions     "transactions orderline inventory"
Route  log       track            logs/log

Route  main      supplant         1
Route  main      receipt          etc/receipt.html
Route  main      master           log mail user
Route  main      cascade          log mail user
Route  main      encrypt_program  "/usr/bin/gpg -e -a r '%s' --batch"
```

This will have the following behavior:

Order

The master order route is *main*, the last one defined. It cascades the routes *log*, *mail*, and *user*, which means they will run in that order at the completion of the *main* route. The individual item routes `HARD` and `SOFT`, if applicable, will run before those.

Transactions

The route *log* specifies the tables that will be put in transaction mode, in this case `transactions orderline`, and `inventory`.

Failure

All order routes must succeed except *user*, which has `error_ok` set to 1.

Encryption The *mail* order route and the `HARD` order route will be sent by email, and encrypted against different GPG key IDs. They will get their `encrypt_program` setting from the main route.

To set the order routing for individual items, some method of determining their status must be made and the `mv_order_route` attribute must be set. This could be set at the time of the item being placed in the basket, or have a database field called `goods_type` set to the appropriate value. The following example uses a Perl routine on the final order form:

```
[perl table=products]
   my %route;
   my $item;
   foreach $item (@{$Items}) {
       my $code = $item->{code};
       my $keycode = $Tag->data('products', 'goods_type', $code);
       $item->{mv_order_route} = $keycode;
   }
   return;
[/perl]
```

Now the individual items are labeled with a `mv_order_route` value which causes their inclusion in the appropriate order routing.

Upon submission of the order form, any item labeled `HARD` will be accumulated and sent to the e−mail address `hardgoods@akopia.com`, where the item will be pulled from inventory and shipped.

Any item labeled `SOFT` will be passed to the order profile `create_download_link`, which will place it in a staging area for customer download. (This would be supported by a link on the receipt, possibly by reading a value set in the profile).

10. SSL Support

Interchange has several features that enable secure ordering via SSL (Secure Sockets Layer). Despite their mystique, SSL servers are actually quite easy to operate. The difference between the standard HTTP server and the SSL HTTPS server, from the standpoint of the user, is only in the encryption and the specification of the URL; `https:` is used for the URL protocol specification instead of the usual http: designation.

IMPORTANT NOTE: Interchange attempts to perform operations securely, but no guarantees or warranties of any kind are made! Since Interchange comes with source code, it is fairly easy to modify the program to create security problems. One way to minimize this possibility is to record digital signatures, using MD5 or PGP or GnuPG, of `interchange`, `interchange.cfg`, and all modules included in Interchange. Check them on a regular basis to ensure they have not been changed.

Interchange uses the `SecureURL` directive to set the base URL for secure transactions, and the `VendURL` directive for normal non−secure transactions. Secure URLs can be enabled for forms through a form action of `[process secure=1]`. An individual page can be displayed via SSL with `[page href=mvstyle_pagename secure=1]`. A certain page can be set to be always secure with the `AlwaysSecure` catalog.cfg directive.

Interchange incorporates additional security for credit card numbers. The field `mv_credit_card_number` will not ever be written to disk.

To enable automated encryption of the credit card information, the directive `CreditCardAuto` needs to be defined as `Yes`. `EncryptProgram` also needs to be defined with some value, one which will, hopefully, encrypt the number. PGP is now recommended above all other encryption program. The entries should look something like:

```
CreditCardAuto   Yes
EncryptProgram   /usr/bin/pgpe −fat −r sales@company.com
```

See `CreditCardAuto` for more information on how to set the form variables.

11. Frequently Asked Questions

11.1. I can't get SQL to work: Undefined subroutine &Vend::Table::DBI::create ...

This probably means one of the following:

No SQL database.

Interchange doesn't include a SQL database. You must select one and install it.

No DBI.

You must install Perl's DBI module before using Interchange with SQL. You can see where to get it at `http://www.cpan.org`, or try:

```
perl −MCPAN −e 'install DBI'
```

No DBD.

You must install the specific Perl DBD module for your database before using Interchange with SQL. You can see where to get it at `http://www.cpan.org`, or try:

```
perl −MCPAN −e 'install DBD::XXXXX'
```

where XXXXX is the name of your module. Some of them are:

```
Adabas
DB2
Informix
Ingres
ODBC
Oracle
Pg
Solid
Sybase
Unify
XBase
mSQL
mysql
```

If you can't make this script run without error:

```
use DBI;
use DBD::XXXXX;
```

Then you don't have one of the above, and Interchange can't use an SQL database until you get one installed.

I don't like the column types that Interchange defines!

They can be changed. See the `foundation/dbconf/mysql` directory for some examples under MySQL.

I change the ASCII file, but the table is not updated. Why?

Interchange writes an empty file `TABLE.sql` (where TABLE is the name of the table). When this is present, Interchange will never update the table from disk.
Also, if you have changed the field names in the file, you must restart the catalog (Apply Changes) before they will be picked up.

Why do I even need an ASCII file?

Interchange wants some source for column names initially. If you don't want to have one, just create a `TABLENAME.sql` file in the `products` directory. For example, if you have this:

```
Database products products.txt dbi:mysql:test_minivend
```

Then create a file `products/products.sql`.
\For:

```
Database pricing pricing.txt dbi:mysql:test_minivend
```

Create a file `products/pricing.sql`. .

Interchange overwrites my predefined table!

Yes, it will if you don't create a file called `TABLENAME.sql`, where `TABLENAME` is the name of the Interchange table. If you want this to happen by default, then set `NoImport TABLENAME`.

11.2. How can I use Interchange with Microsoft Access?

Though Interchange has ODBC capability, the Microsoft Access ODBC driver is not a network driver. You cannot access it on a PC from your ISP or UNIX system.

However, you can turn it around. Once you have created a MySQL or other SQL database on the UNIX machine, you may then obtain the Windows ODBC driver for the database (MySQL has a package called myODBC) and use the UNIX database as a data source for your PC−based database program.

Here is a quick procedure that might get you started:

- Get MySQL from:

  ```
  http://www.mysql.com/
  ```

Install it on your UNIX box. On LINUX, it is as easy as getting the RPM distribution:

```
http://www.mysql.com/rpm/
```

You install it by typing, as root, `rpm −i mysql−3.XX.XX.rpm`. If you are not root, you will have to build the source distribution.

- To avoid permissions problems for your testing, stop the MySQL daemon and allow global read−write access with:

  ```
  mysqladmin shutdown
  ```

```
safe_mysqld --skip-grant-tables &
```

Obviously, you will want to study MySQL permissions and set up some security pretty quickly. It has excellent capability in that area, and the FAQ will help you get over the hurdles.

- Set up a database for testing on the UNIX machine:

```
mysqladmin create test_odbc
mysql test_odbc
```

Make an SQL query to set up a table, for example:

```
mysql> create table test_me ( code char(20), testdata char(20) );
Query OK, 0 rows affected (0.29 sec)

mysql> insert into test_me VALUES ('key1', 'data1');
Query OK, 1 rows affected (0.00 sec)

mysql> insert into test_me VALUES ('key2', 'data2');
Query OK, 1 rows affected (0.00 sec)

mysql>
```

- Get and install myODBC, also from the MySQL site:

```
http://www.mysql.com/
```

You install this package on your Windows 95 or NT box. It is a simple setup.exe process which leads you to the control panel for setting up an ODBC data source. Set up a data source named `test_odbc` that points to the database `test_odbc` on the UNIX box. You will need to know the host name and the port (usually 3306).

- With Microsoft Access, you can then open a blank database and select: File/Get External Data/Link Tables. Select File Type of 'ODBC databases' and the proper data source, and you should have access to the database residing on the UNIX side.
