This Document

This document discusses Morris Templating Language (MTL) classifieds tags: their uses, usage and syntax.

It refers to concepts discussed in "MTL Basic Concepts" and in "MTL Common Tags." If you haven't already read these documents, you are encouraged to do so.

Conventions

This document will show sample results as they would appear in a browser.

- In commentary text, data tags will be discussed in terms of what comes after the second dot. For example `<mcc classifieds.ad.id>` will be referred to as "id" or "the id tag." In example code, the tags will be fully written out as they should appear in a template.

  Query qualifiers, on the other hand, will always appear as they should in working code; there are no "other parts" to a query qualifier. A qualifier described as "id" in commentary should be written "`id=`" in an actual query.
- "Template," as it will be used in this text, can mean two things: [1)] a document in Template Manager (TM) and its contents or [2)] a section of MTL code that (should) stand alone and work without other supporting MTL code.
- Anything appearing in square brackets: [like this] is just a place holder. Do not insert the square brackets and text inside into a template; replace it with appropriate values.
- `Code to be typed into a template will look like this.`
- Result text (template output) will look like this

Miscellaneous Notes

- This document is not intended to be an exhaustive tag reference. Some sections cover every query qualifier and data tag available in a tag set (at this writing,) others do not. For a full tag reference consult the "MTL Tag Quick Reference."
- Line breaks may be inserted in the middle of MTL tags for clarity throughout this document, but line breaks are NOT LEGAL inside MTL tags*. If you copy the code in this document into templates, REMOVE ALL LINE BREAKS that appear between instances of "<mcc" and ">".
- None of the queries that will be covered in this document will execute without at least one qualifier. Because the result sets retrieved by classifieds queries are generally very large, an automatic brake has been applied: every query must be qualified by at least one value. And qualifiers with blank variables on the other side of the equals (e.g. `id=$id`, but `$id=''`) don't count.

About the Author

G. Ann Campbell has been writing MTL since 1999 - when she started by timidly changing the table cell background colors on onlineathens.com's classifieds templates.

In 2001 she joined MDW and began taking on larger MTL projects, such as the construction of the calendar template set deployed to most Morris Communications Corporation Web sites, and building classifieds template sets from scratch.

Since then her mission has been to explore strange new tag sets, to seek new uses and new output formats. To boldly go where no MTL-coder has gone before.

* Except in the values of variables. It's perfectly okay to put a linebreak inside a variable - but we won't be doing that in this text.

Table of Contents

Majors & Minors

Classifieds are generally divided into broad categories – majors – like Employment and Transportation, and under those majors into narrower groups – minors – like Ford or Saturn.

In TM you'll see these two divisions labeled "class major" and "class minor." A major category is the top category and a minor category, where the ads are actually kept, is the lowest level. Any levels in between are called "sub-majors."

In the MTL tags you'll see the hierarchy treated a little differently.

The MTL classification hierarchy is made up of several levels of categories. Each category has a name. Categories at the lowest level, where the ads are stored, also have category numbers. (Usually this is the classification number from the property's print side or ad entry system.)

In MTL the term "category" refers to a single category while, the term "classification" refers to a category *and* all of the categories below it. So "Classifieds," which is generally used to refer to Classifieds *and* all the categories below it, is generally thought of in the "classification" sense.

Similarly, majors (and sub-majors) are usually used to refer to a major itself and all its minors. So they're generally referred to in the classification sense. But majors and sub-majors can also be considered "categories." I.e. you can talk about the "Classifieds" category - when you want to refer to Classifieds alone, without any of the templates or other categories under it.

Only the categories with numbers contain ads, but categories at every level can contain templates.

Standard Majors (and sub-majors)

A standard set of class majors is used in all MTL-driven classifieds. There are two main benefits of the system: aggregation and parsing.

As ads at the local property are uploaded to the central database MTL-driven content is served from, ads going into certain majors are parsed for details. Specifically, if it wasn't sent with the ad, ads in the Real Estate major are parsed for the number of bedrooms, bathrooms and price. Similarly, ads in the Transportation major are parsed for make, model, year and price.

Having standard majors also means you can easily combine your classified ads with those of your nearest neighbors to expand your car/truck listings from local to regional.

Does that mean you have to abandon the class majors your paper uses to put your classifieds online? Not at all. The MDW system is flexible enough to let you use the standard majors and also maintain the majors (as sub-majors) your paper uses. In fact, even under the standard set of majors, sub-majors are needed under Real Estate: Real Estate for Rent and Real Estate for Sale.

Here's the standard setup:

ANNOUNCEMENTS
BUSINESS
EMPLOYMENT
LEGALS
MERCHANDISE
REAL ESTATE
  REAL ESTATE FOR RENT
  REAL ESTATE FOR SALE
SERVICES
TRANSPORTATION

Configuring a template

Each TM property can have one classifieds category. It will be named "CLASSIFIEDS." If your TM property doesn't have a classifieds category, contact your Support representative.

All your classifieds templates will be in the classifieds category. (Either directly under Classifieds, or under a major or sub-major.) There are five configurable attributes for each template: status, type, sub-type, default and level.

- The type of all the templates under classifieds should be "classified."
- Status should be "active" for any live templates.
- Default should be set to true for production templates (you can use "false" for templates you're testing.)
- The sub-type of a template should vary according to its function. Sub-types of note are:

  `TOC` Table of Contents - Generally this will be a (linked) listing of your majors and minors. Generally there will be one TOC per property.

  `Search` A search form/interface

  `Detail` Ad display

  `Browse` Ad display

  `Alt_Detail` Ad display - This is often used for alternate presentation, e.g. a printer friendly version

  `Clipboard` This sub-type is used in conjunction with the 'My List' feature to display the ads the user selected

  `Include` You **must** use this sub-type for code (MTL/HTML/plain text) you'll be including into other templates. You cannot include a template that is not of this type

- Set level to1 for the templates you expect to use the most; it's the default level. The original intent of the level input was to distinguish among multi-page interfaces, like data entry forms or iterative search (search within results.) So page 1 of an interface might be at level 1, and the second page at level 2 &etc.

Template organization

Before we talk about MTL, we need to talk about template organization.

It would be entirely possible to write a single template that would handle every function in your classifieds: search interface, results, highlighted ads &etc. But printed out, with the pages laid end-to-end, it would stretch about a mile, and the thought of changing a link color would give you a migraine.

So it makes sense to spread the function out among several templates: a search template, a results (detail) template, a highlighted ads (clipboard) template &etc.

Let's say you have these three, self-contained templates. It makes life much easier … until it's time to change the static links in your classifieds. Now, because these links show up throughout your classifieds sections, instead of editing one template (the mile-long one) you have to edit three. Still not an entirely satisfactory situation.

But if you stripped the HTML out of your templates up to the point MTL generated content begins and compared the code you got, you'd probably find that they're pretty much the same. So consolidate them into one place: make a new template - an include named "header." Then include it into your other three templates. This way, the next time you have to change the links in the header, you have one template to edit and all your output gets changed. Same with the footer code.

Now, under Classifieds, we have five templates: a header include, a footer include, a detail, a search and a clipboard. Presumably, the detail, search and clipboard all include the header and footer.

But you already know that Real Estate and Transportation have extra data available - certain features get parsed out of those ads. And because this extra data is available, you'll want to let your users search it - so that means extra inputs in the search template. You'll probably want to display it too, and that means extra outputs in the detail.

You could handle these "special cases" in your search and detail templates with if statements. Or, you could create special templates under those classifications - which is what I recommend.

For instance, Real Estate detail templates typically display the numbers of bedrooms and bathrooms separately from the main text of the ad - generally they get their own columns in the HTML table structure. So do you build those extra columns and extra outputs into the main detail with if statements? Having tried it that way, I can say it's a lot easier to have one template (under Real Estate) where you *always* output those things and one (directly under Classifieds) where you *never* do.

So that expands our number of templates to nine: search, Real Estate's search, Transportation's search, detail, Real Estate's detail, Transportation's detail, clipboard, header, footer. (If you're doing alternate presentations, like a printer-friendly version, you'll need three of those too.)

Is that a lot of templates to manage? Yes. Is it too many? Depends on your point of view. Generally, my philosophy is to consolidate what can be consolidated - don't repeat your html head, style sheets and javascripts in every template; put them in one place for centralized management. That not only makes your life easier at redesign time, but has an added benefit: it keeps your other templates stripped down to just the MTL and leaves you a lot less to wade through when you need to make functional changes.

On the other hand, consolidation can be over-done. Just because all three of your detail templates use a couple of the same data tags doesn't necessarily make them candidates for consolidation.

Also, as the number of templates escalates, you'll find yourself spending a lot of time trying to figure out which template you're looking for - unless you adopt a consistent (yes, this actually means "boring") name-scheme.

I recommend naming templates after their functions: name your main detail template "detail," and your search template "search." For detail and search templates not directly under Classifieds, like the ones under Real Estate, I'd still name them "detail" and "search," or maybe "rdetail" and "rsearch" (so if you have the main detail and the Real Estate detail open at the same time, you can tell the difference between them in the title bar.)

Also, I strongly recommend putting the word "include" into the names of include templates, so "header" becomes "header include." Or even better, name them all like this: "include - header" and they'll group together in your template list and be easy to find.

Obviously, these are matters of personal preference and style, and it can be a fine line between "split this into another template" and "just wrap an if statement around it." But as you build your classifieds template set, you should give serious consideration not just to your current convenience in building the structure, but to your future convenience in maintaining the templates.

Accessing a template

A dedicated second or third level domain is required for classifieds. Typically a third level domain is used. They often take the form "classifieds.[something here].com," but don't have to. If you're not sure what yours is, contact your Support rep.

The URL to access your templates will follow this pattern:

```
http://[domain]/[RPE]?classification=classifieds&temp_type=[your sub-type]
```

RPE stands for "request processing engine," a generic name for the programs that take requests from the Web, process and respond to them. There are a few slightly different versions: "classifieds-bin/classifieds," "realestate" and "Forge." What's the difference? Not much. They're all names for basically the same thing. Using the classifieds-bin/classifieds name triggers a slightly different behavior in some situations. (We'll go into specifics later.) As you might guess, realestate's a little more tuned to the real estate vertical – an outgrowth of classifieds. Classifieds-bin/classifieds' special behaviors are a little more classifieds-centric.* Since classifieds is the focus of this document, we'll use it for most of our examples. We'll refer to the basic program that all three are versions of as "the RPE."

So in the example above, if the temp_type in your URL is "detail," you'll get back the active, default, level 1 detail template directly under the specified classification, classifieds.

In any category (including classifieds) you can only have one template of a given subtype and level set to status=active, default=true. (i.e. only one default, active, level 1 detail template in a category.) When you need a second template of a given sub-type, set level to 2 and add "&tl=2" (for template level) to your URL:

```
http://[domain]/classifieds-bin/classifieds?temp_type=[sub-type]&tl=2
```

While you can't have multiple default, level 1, detail templates directly under classifieds, you can have one under Classifieds and one under each major and sub-major under Classifieds.

When you use the URL format above, you get the default, level 1 detail template in classifieds. If you want to get to the default detail template in one of your majors, add it to the URL, like so:

```
http://[domain]/classifieds-bin/classifieds?temp_type=detail&classification=real+estate+for+sale
```

(Note the use of pluses (+) for spaces.)

The classification you specify tells the RPE where to start looking for a template that meets the temp_type and level requested. If it doesn't find one there, it'll go to the next level up and look there (and so on.)

The first URL's in this chapter, told the RPE to start looking at the highest level, classifieds. The URL directly above, tells the RPE to start looking for a default detail (active, level 1) in Real Estate for Sale. If it doesn't find one, it'll look at the next level up: Real Estate, then up again to Classifieds. If there isn't one in Classifieds, you get a "template not found" error.

Of course, that begs the question: If you don't have a default, active, level 1 detail template in Real Estate for Sale, what's the point of putting the classification in the URL? It's used to indicate that you want data from that classification (more on this later.)

When your property's classifieds domain was set up, certain associations were made for it. For instance, the classifieds.onlineathens.com domain is associated with the "Athens Ga" template property in TM. So when you go to:

```
http://classifieds.onlineathens.com/classifieds-
bin/classifieds?temp_type=toc
```

You'll get the default, level 1, TOC template in the Athens Ga property. But if you'd like the default, level 1, TOC template from another TM template property, you can still use the onlineathens domain:

```
http://classifieds.onlineathens.com/classifieds-
bin/classifieds?temp_type=toc&tp=augusta+ga
```

Adding "`tp=[template property]`" switches the Template Property used for the request from the default property associated with the hostname to the one specified in the URL. Of course, this is mainly for testing; you don't want to deploy links to another site's third level domain: aside from the branding issues, the other site will get credit for your hits.

Exercises:

Ex. 1: Create a template somewhere in your classifieds tree. Don't worry about MTL yet, just have it say "Hello Classifieds." Navigate to it. Do not use "tid=" in the URL.

* All templates can be triggered from the command line to output static documents. (For more on this see *Fileability* in "MTL Common Tags.") Despite the differences between the two RPE's, there's only one command line version of the program: Forge. The differences between the two RPE's are Web request-specific, and command-line Forge is fully capable of performing all other classifieds functions.

Property

We've seen that your template property is automatically associated with your property's classifieds hostname, but can be overridden by using "`tp=…`" in the URL.

Similarly, your data property (which set of ads to use) is also automatically associated with the hostname.

So when you go here:

```
http://[domain]/classifieds-
bin/classifieds?classification=classifieds&temp_type=detail
```

several variables are set for you. Obviously, `$classification` and `$temp_type` are set from the URL, but behind the scenes, a few other variables get set:

```
$tp = [your template property]
$property = [your data property]
```

If you were to print out these two variables, they'd look remarkably similar. That's because they're the same. Your ads are associated with the same TM property your templates are stored in (even though you can't see the ads from TM,) so your data property has the same (case insensitive) name as your template property.

Both `$tp` and `$property` are obtained by using the hostname UNLESS one or both are specified in the URL (using `property=` and `tp=`.) If `$tp` is set in the URL and `$property` is not, `$property` is assumed to be the same as `$tp`. If `$property` is specified and `$tp` is not, the value for `$tp` is obtained by using the hostname. If `$tp` or `$property` values cannot be obtained from using the URL and are not specified explicitly, you get a lovely message (in the case of missing `$tp`) or you get no ads (in the case of a bad `$property.`)

So what's `$property` for?

Each classifieds query has a "`property=`" qualifier. It tells the RPE which property's data to use. So "`$property`" is what goes on the right side of that equals sign.

But obviously, your property is the one you're writing the templates for, so why bother with `$property`? It makes more sense to write all your queries like this, right?:

```
<mtl classifieds.[table] property=[myProperty] … >
```

Not exactly.

One of the underlying goals of the classifieds structure is to make aggregation of ads from different properties easy. And hard-coding the name of your property into all your queries stumps that.

So do it this way:

```
<mtl classifieds.[table] property=$property … >
```

And if you want to aggregate, pass "`&property=[myProperty],[anotherProperty]`" in through the URL.

Even if you don't care a jot for aggregation today, use `$property` in your queries anyway. Because someday you, or your boss(es) or your successor(s) will.

At this point some of you are wondering "if the RPE is so smart, why do I even have to specify `property=$property` in the query?"

The answer has two parts:

1. "Assuming" property (instead of having it specified) would make aggregation difficult.

2. The RPE is good, but it's not that good.

Classifieds.Category

Now that you've got `$property` and templates established, we'll start looking at what's under your property. The classifieds.category query will give you a list of all the majors, sub-majors and minors below a given level.

Often the "given level" is classifieds, but it doesn't have to be. Here's a brief rundown of the **qualifiers** specific to classifieds.category:

- `classification=` Use this qualifier to show every "category" from the given point (inclusive) down; specify "classification=classifieds" to get classifieds itself and every major, every sub-major and every minor.
- `number=` The category number (or comma-delimited list of numbers) of a given minor. Since there will be no other categories under a minor, you'd use this to get info about that specific minor.
- `parent_id=` Use this qualifier to show every "category" (non-inclusive) directly under the given point. Specify parent_id=[the unique id of your "classifieds" category] to get a list of your majors. (Note, this is also a valid orderby value.)
- `show_ancestors=` The argument for this qualifier is a whole number. It specifies how many levels to show "up" from the specified classification. (Although, if your classification is Classifieds, you don't want to go much higher.)
- `show_children=` The argument for this qualifier is a whole number. It specifies how many levels to show "down" from the specified classification. (But if your classification is a minor, you can't get much lower.)

With the show_[family] qualifiers, there are a few things to keep in mind: Don't use them together; they contradict each other. (I.e. You can only go in one direction.) Also they both number the starting point as "1," so show_[family]=1 will return only the starting point.

Now, a brief look at some of the **data tags**:

- category_id - The unique id of the current major, sub-major or minor.
- images_url - This is the URL to your classifieds logos directory. This was set up for you when your classifieds were set up. If you have questions about this, contact your Support representative.
- level - Unless your query is qualified by 'show_ancestors=' or 'show_children=' this will always return "1". However if you are using one of those two qualifiers, it shows the level of the current record relative to the starting point, e.g. if you started at Classifieds, level returns "2" for all your majors.
- name - The name of the current record.
- nav_url - This is the base URL to classifieds, i.e. http://[domain]/classifieds-bin/classifieds associated with the property of the current record. (Think "aggregation.")

- number - The category number of the current record. This will be blank for Classifieds itself, majors and sub-majors. (Only "minors" have numbers.)
- parent_category - The name of the level directly above the current one. For majors, this will be "classifieds."
- parent_id - The category id of the level directly above the current one.
- property - The property the current record is from (think "aggregation.")
- property_id - The unique id of the property the current record is from.

All that said, how do you pull out a list of your majors? There are a couple of ways to skin that cat, but the most straightforward is probably this:

```
<mtl classifieds.category property=$property parent_id=[unique id of your
classifieds category]>
     <mtl classifieds.category.name><br>
</mtl classifieds.category>
```

The problem with this code is that it only returns the records directly under Classifieds. So you get all your majors, but none of your sub-majors.

Let's try again:

```
<mtl classifieds.category property=$property classification=classifieds>
     <mtl if {classifieds.category.number=''}>
          <mtl classifieds.category.name case=title><br>
     </mtl if>
</mtl classifieds.category>
```

Notice the if statement, which filters out the minors. (This and subsequent examples in this chapter use the Demo classifieds tree.) The browser output looks like this:

Classifieds
Legals
Employment
Announcements
Real Estate
Real Estate for Rent
Real Estate for Sale
Merchandise
Antiques and Collectibles
Yard Sales and Auctions
Computers
Pets
Recreational
Agricultural
Business
Financial
Education
Services
Transportation
Automotive

Motorcycles
Boats and Planes
Auto Services
Obituaries

There are, of course, a few problems with it. I'd like to suppress Classifieds itself, as well as the majors that have sub-majors. And it would be nice to have the results in alphabetical order.

Unfortunately, the only way to suppress the records I don't want is to filter them out with an if statement. In Demo, the majors with sub-majors are Announcements, Business, Merchandise, Real Estate and Transportation. So here goes:

```
<mtl classifieds.category property=$property classification=classifieds
orderby=name>
     <mtl if {classifieds.category.number=''}>
     <mtl if {
          classifieds.category.name<>"Classifieds" and
          classifieds.category.name<>"Announcements" and
          classifieds.category.name<>"Business" and
          classifieds.category.name<>"Merchandise" and
          classifieds.category.name<>"Real Estate" and
          classifieds.category.name<>"Transportation"}>
               <mtl classifieds.category.name case=title><br>
     </mtl if>
     </mtl if>
</mtl classifieds.category>
```

Notes:

- This could have been accomplished with one large if statement, but was done in two for (slightly enhanced) clarity.
- Research your tree before using this code. Other than Real Estate and Classifieds, the majors you want to suppress will likely be different.

The code above returns this output:

Agricultural
Antiques and Collectibles
Auto Services
Automotive
Boats and Planes
Computers
Education
Employment
Financial
Legals
Motorcycles
Obituaries
Pets
Real Estate for Rent
Real Estate for Sale

Recreational
Services
Yard Sales and Auctions

On the other hand…

Looking at the Demo tree, I see that Computers, Pets, Recreational and Yard Sales are all minors of the Merchandise major I suppressed, but rather than being grouped together, they're scattered through the listing. So maybe what I want is a hierarchical view :

```
<mtl classifieds.category property=$property classification=classifieds
show_children=3>
     <mtl if {classifieds.category.number=''}>
          <mtl if {classifieds.category.level=3}> - </mtl if>
          <mtl classifieds.category.name case=title><br>
     </mtl if>
</mtl classifieds.category>
```

This time we get:

Classifieds
Legals
Employment
Announcements
- Obituaries
Real Estate
- Real Estate for Rent
- Real Estate for Sale
Merchandise
- Antiques and Collectibles
- Yard Sales and Auctions
- Computers
- Pets
- Recreational
- Agricultural
Business
- Financial
- Education
Services
Transportation
- Automotive
- Motorcycles
- Boats and Planes
- Auto Services

There are a few things to note about this example:

- In the previous example I used multiple if tests for clarity, but here they're separated for functionality.
- I used the show_children qualifier so the level data tag would return something meaningful … and it was the level tag that let me show hierarchy.

- Show_children has to be set to 3 for the query to get "down" to the sub-majors. (I.e. Our starting point, classifieds itself, is at level 1, the majors are at level 2 and the sub-majors are at, you guessed it, level 3.)
- Show_children (and omitting and orderby qualifier) is necessary if you want your results to be returned hierarchically, i.e. each major with it's sub-majors and minors directly underneath. Otherwise, they'll come back in an arbitrary order.
- I still had to suppress the output of the minors. In this case, minors under sub-majors are at level 4 - and wouldn't be returned by our query. But minors directly under majors are right next to sub-majors, at level 3.
- Classifieds still shows up and would have to be suppressed with an if statement if I didn't want it.

Exercises:

Ex. 1: Directly under your Classifieds category, create a TOC template that shows your majors and/or sub-majors.

Ex. 2: Also under the Classifieds category, make another TOC template to show the minors under a major/sub-major. Modify the template from Ex. 1 to link each major to this template, passing in the major/sub-major this template will use. Make sure this template honors that passed-in value.

Ex. 3: Directly under your Classifieds category, write a search template that creates a form. The form should submit to classifieds-bin/classifieds, and give the user a way to choose one of your majors/sub-majors and/or minors. (For more on writing forms, see *Forming* in "Common MTL Tags.")

Classifieds.Ad - basic query

Of course, pulling out lists of majors and minors is all well and good, but classifieds is about ads, right?

Right. And you do that with a classifieds.ad query, something like this:

```
<mtl classifieds.ad property=$property>
     <mtl classifieds.ad.body><p>
</mtl classifieds.ad>
```

From that query you get back the basic text of the first 20 ads from the specified property.

Here we use the most important **data tag** in classifieds:

body - This is the text of the ad. The reason for the whole shebang.

> Depending on the front-end system it came from, it may or may not already contain some HTML formatting: some properties bold the first three words of an ad automatically. Others upsell bolding the whole ad - then pass that formatting forward to the online listings. Be aware, though, that they don't always put in their close tags like they should. If an entire ad, and everything after it, shows up bold, you've probably got an ad missing it's .
>
> Some ads may also contain line breaks. If you'd like to display them online, add `linebreaks="<br>"` to your data tag, like so:
>
> ```
> <mcc classifieds.ad.body linebreaks="<br>">
> ```

But, that begs a few questions: How do you get to the next 20 ads, and how do you tell the RPE which 20 ads to start with?

These **qualifiers** let you specify the major or minor to pull from:

- `property=` The first criteria is which property's tree to look in. It should appear in every classifieds.ad query you write.
- `category=` This is the name of the minor you're interested in.
- `category_id=` The unique id of the minor you're interested in.
- `category_number=` The number of the minor you're interested in (corresponds to `<mtl classifieds.category.number>`).
- `classification=` The major/sub-major you're interested in - and all the categories below it.
- `classification_not_in=` Major(s) and the categories underneath to exclude from results.

Similarly, you can find out about the category and classification of an ad with these **data tags**:

- category - The ad's minor.
- category_id - The unique id of the ad's minor.
- category_number - The number of the ad's minor.
- parent_category - The name of the category above the ad's minor - This will be ad's major or sub-major.
- parent_id - The unique id of the ad's major or sub-major.

These **qualifiers** let you get to subsequent pages of ads:

- `page=` This qualifier takes a whole number specifying which page of ads to show.

  This qualifier is actually optional: if your URL contains "page=", the numeric value to the right will be automatically be applied. Specifically, even if your query omits "page=$page," but your URL contains "&page=3" you'll get the third page of ads. (Note: This is true ONLY in classifieds.)

  In some cases you'll need to override this behavior. You can do so by hard-coding the page number you want in the query, e.g. "page=1".

- `ads_per_page=` How many ads to show on each page. Leave this out and you'll get the default: 20. If you set this to a number higher than 100, it will be ignored and again you'll get the default: 20. **This automatic pagination behavior is unique to the classifieds.ad tag set.** It appears here because the record sets returned by classifieds.ad queries are usually very large. Automatic pagination appears nowhere else in MTL.
- `nomax=` If, for some reason, you don't want your ads returned in controlled chunks, use `nomax=true` in your query. The RPE will return every ad that matches your query at one time, rather than breaking it into pages. This qualifier overrides ads_per_page - both the built-in default and any value you set explicitly. E.g. if your query finds 632 ads, and your query contains `nomax=true`, you'll get all 632 ads on page 1 - even if you're also using `ads_per_page=` . So use this cautiously - you may get very large result pages with it.

These pagination **data tags** are unique to classifieds.ad. For more on pagination, see *Pagination* in "MTL Common Tags.":

- adcount - This is the classifieds.ad version of the num_rows data tag found in other tag sets.
- range_min - This is the number of the first ad on the page - relative to the number of ads found. Let's say 300 ads match a search. But they're presented in pages of 20. On the first page this tag returns "1". On the second page this tag returns "21." On the third page it's "41" &etc.

- range_max - This is the number of the last ad on the page - relative to the number of ads found. For ads returned in pages of 20, the values returned by this tag would be 20, then 40, then 60 &etc.

So let's take another look at that query:

```
<mtl if {$maxrec=''}>
     <mtl $maxrec=25>
</mtl if>
<mtl if {$classification=''}>
     <mtl $classification="classifieds">
</mtl if>

<mtl classifieds.ad property=$property ads_per_page=$maxrec page=$page
     category_number=$categorynumber classification=$classification>

     <mtl if {classifieds.ad.index=1}>
        <div align=center>Displaying
              <mtl classifieds.ad.range_min> -
              <mtl classifieds.ad.range_max>
              of <mtl classifieds.ad.adcount> ads found.<br>
              <mtl classifieds.ad.page_nav_bar></div><p>
     </mtl if>

     <mtl classifieds.ad.body><p>

</mtl classifieds.ad>
```

Let's look at the **qualifiers** in use:

- `ads_per_page=$maxrec` - Why not hard-code this? Sometimes you want to let the user specify this - so I used a variable in the query. If the variable is undefined (if there's nothing in it,) you'll get the default value. So the if statement above the query sets the variable to my (non-default) value if it's blank.
- `classification=$classification` - I alluded to this qualifier in *Accessing a template* - Unless you want to write a new version of your template for each major in your classifieds (and I'm sure you don't) use a variable here, but set a default value incase a value's not passed in for this.
- `category_number=$categorynumber` -Typically, `$categorynumber` (or whatever you choose to name your variable) will have been chosen by the user from a form. If this value is blank, you'll get every ad in $classification.

Exercises:

Ex. 1: Directly under your Classifieds category, create a detail template to show ads from a given class major or minor. Be sure to assign a value to $classification if one isn't passed in.

Ex. 2: Modify your form from Ex. 3 in Classifieds.Category to submit to your template from Ex. 1. Let the user choose the number of ads per page and either the major or the minor. Make sure the template from Ex. 1 honors the choices submitted from this form.

Keywords

You should at this point have a detail template and a search template that submits to it. Of course, having a "search" template implies keyword search.

Here are the **qualifiers** to implement that:

- `keyword_query=` This is the only qualifier you actually need to implement keyword searches. Obviously, you'll want a variable on the right side of this equals sign. Hard-coding the keyword would kinda defeat the purpose, right?
- `keyword_font=` Use this qualifier to highlight occurrences of the keyword(s) in the results. The syntax is this: `keyword_font='[open tags],[close tags]'`. Notice that the opening and closing tags are separated with a comma. Everything to the left of the comma goes before the keyword, and everything to the right goes after. Do not use commas in either the open or close tags themselves; you will get bizarre results. All together, your open tags, close tags and the comma between them may add up to no more than 64 characters. Here's a suggested (37-character) use:

```
keyword_font="<font color='#cc3333'><b>,</b></font>"
```

- `keyword_search_logic=` Leave this out of your query and you'll get the 'AND' behavior. There are several search logic options. You can hard-code one of the choices into your template, or give your users the logic option via a form input.

  `and` You get results that contain the first keyword **and** the second keyword **and** the third…

  `or` Set keyword_search_logic=or and you'll get results that contain the first keyword **or** the second…

  `fuzzy` The fuzzy operator expands the list of words to include all words with similar spelling. For example, "read" expands into "lead" and "real."

  `about` Use the About logic for contextual searches. It expands the list of keywords to retrieve ads that contain concepts that are related to your keyword or phrase. For example, if you your search term is "sports and entertainment" the About operator will give you ads with terms like "equestrian," or "race" in addition to bringing back ads with "sports" or "entertainment" in them.

  `soundex` Returns matches where the word sounds like the one you typed (i.e. ROAD would return ROAD, RODE, etc.)

  Note that this option may have a significant slow the speed at which your results come back.

  `stemming` Stemming enables you to match words with the same linguistic root (i.e. scream expands to: scream screaming screamed. However, carp will not be expanded to carpet - different linguistic root.)

- `search_score=` This specifies a minimum search score to display. Ads with search scores below this number will not be returned. For more on this (and before using it) please see the description of search scores below.

The only **data tag** associated with keyword searching is

```
<mtl classifieds.ad.search_score>
```

However, you're not likely to find this very useful. The number you get back is a calculation, and while you might want to order your results by search_score you probably won't want to print out the number.

Results are scored using an inverse frequency algorithm based on Salton's formula. The scoring assumes that frequently occurring terms in a record set (like "the") are 'noise' terms, so they get scored lower. For an ad to score high, the query term must occur frequently in the ad, but infrequently in the set of ads as a whole*.

In other words, the numbers you get back aren't usually going to be the kind of relevance scores (i.e. 99% out of 100%) people are used to seeing. The highest search score I've seen for an ad - one with all my search terms - has been 35.

Exercises:

Ex. 1: Implement keywords in your detail template from *Classifieds.Ad - basic query*, Ex.1.

Ex. 2: Modify your search template from *Classifieds.Ad - basic query*, Ex. 2 to give the user a keyword search option and a choice on the search logic. Make sure the names of your inputs match up with the variable name you used for Ex. 1.

* This description of search scores is from "Oracle8i interMedia Text Reference": http://desktop.msfc.nasa.gov/oracle815doc/inter.815/a67843/ascore.htm#272. Results may differ depending on the database management system in use.

The Lifecycle of a Liner

At this point it's useful to talk about the lifecycle of a typical classified ad. The details of this lifecycle may vary from company to company, but the general outline is as follows:

Most ads start life at the local paper's Classifieds Department, where they're typed into and scheduled through the local front-end system. For the sake of discussion, let's say we're talking about a Real Estate ad:

```
5.2 Acres. 4BR, 2BA, LR, DR,
den, hardwood & tile, CHAC.
    Fenced. Only $99,500
       Call 555-1212
```

It'll start on the 20th and run through the 5th of next month. Today happens to be the 15th.

Tonight, after the classifieds deadline, the classifieds admin will export ads from the front-end system and upload them to be parsed and inserted into the database. Some systems (e.g. DTI) will send this ad only once - marked with the start date and end date, and they'll do it tonight. Others (e.g. Baseview's AdManager Pro) will hold it until the night before it starts, the 19th in our case. They'll send it nightly, without start and end dates, every night until the end date. Each day's copy of our ad will be live for a single day.

In either case, the ad is uploaded **before** it's scheduled to go live.

Shortly after the ads are uploaded, they're picked up by a parser that examines each Real Estate and Transportation ad for certain features, then inserts the ads and their features into the database. When this ad is parsed, it'll pick up on the fact that there are 4 bedrooms, 2 bathrooms, 5.2 acres and that the price is $99,500.

The ad will be inserted around 8 p.m., Eastern time either on the 15th or the 19th, status=active. (The timing of this and several other events in this lifecycle are implementation-specific. In other words, it's one way to do it, but not the only way. If you have questions about how it's done at your company, contact your support team. Subsequent instances will be marked with an asterisk [*])

- For ads from DTI, start date will be set to the 20th, end date to the 5th.
- For AdManager Pro ads sent from a daily publication, start date will be the 20th, end date will be the 20th, and tomorrow it'll be reinserted, start date=21st, end date=21st, and so on.
- For AdManager Pro ads sent from a non-daily publication, start date will be set to the 20th, but end date will be adjusted based on the STOP_DATE_OFFSET property attribute set in TM. For a weekly publication, this attribute would have a value of 7, and the end date of our ad would be adjusted to the 20th + 7, or the 27th. Then, next week it would be inserted for another seven days, and so on.

We'll call the period from an ad's start date to its end date is its "run span."

It is important to note that templates not qualified by date (more on this soon) may pick up this ad prematurely at this point: 5 days prematurely for some, just a few hours prematurely for others.

At 12:00 a.m. in the time zone the ads are from (as specified by the TIME_ZONE property attribute in TM) on the 20th, this ad officially goes "live."

Most ads stay live in the system until about 4 a.m. on the day after their end dates. Our example ad would be changed to status=expired around 4 a.m. Eastern on the 6th*.

Some ads, however, will be "killed" during their run spans. For example, if a deal closed on the house in our example on the 28th, we'd want to stop getting calls; we'd want the ad killed.

In that case, the status of the ad would be changed from "active" to "killed." No other piece of info about the ad would be edited; the end date remains the 5th. And at 4 a.m. on the 6th, this ad's status would **not** be changed to "inactive." This ad would retain the killed status to indicate that it went "off-line" prematurely.

In any case, whatever an ad's status, it will remain in the classifieds database for eight days* after its end date. (One reason is so template writers can let users "Search Today and Sunday," for instance.)

Around 12:30 a.m. Eastern on the 9th day* after they expire, ads are either archived or thrown away. Whether an ad is archived or not depends on whether the category the ad is in has the "Archive" checkbox checked in Template Manager. (Edit a category in TM to see this checkbox or change its value.)

Our example ad will be archived – moved from the classifieds database to an archive database – or dumped on the 14th. At this point it's no longer available through MTL.

*Implementation-specific

Date, Status

When you run a classifieds.ad query, by default it returns ads that are active or pending.

That typically translates to getting ads that aren't yet expired; i.e. today's date falls between the ad's start date and end date, but as we discussed in *The Lifecycle of a Liner* above, we could also prematurely get ads that aren't supposed to be live yet.

And that might not be what we want.

To get ads of a different status, use the `status=` **qualifier**. Leave this out and you'll get the default, "active,pending" which is generally what you'll want. Valid statuses are:

`active` Ads whose start_date has passed (or is today) and whose end_date has not come yet are active

`expired` When an ad reaches the end of its run, its status is automatically changed to expired.

`killed` Ads of this status were generally pulled before their time - usually at the request of the customer. E.g. "I've sold my lamp, now I want people to stop calling me. I know I paid for it to run the rest of the week, but KILL IT NOW!"

`inactive` This status is applied to some ads that are entered from the Web. The status of those ads is "inactive" while the system waits for confirmation of payment. This status is also used for ads in the Real Estate Vertical entered from a Multi-Listing Service system.

`pending` This status is used for ads in the Real Estate Vertical entered from an MLS system.

`pending approval` This status is applied to some ads that are entered from the Web. The status of those ads is "pending approval" after confirmation of payment has been received but before the ad has received editorial approval.

To get an ad's status, use the status **data tag**:

```
<mtl classifieds.ad.status>
```

The status qualifier and data tag always apply to an ad's **current** status. No records are kept of an ad's status history. So querying for `status=active date=yesterday` **will not** get you ads that were active yesterday. It will get you ads that are currently active and where yesterday's date is within the ad's run span.

To pick ads based on date, you can use the following **qualifiers**. All of them accept dates in any of the standard formats MTL recognizes (for more see *Time, Time, Time* in "MTL Common tags"):

- `date=` Use this qualifier to pull back ads where the specified date is within the ad's run span.
- `date_created=` Use this qualifier to find ads created (added to the database) on a given date.

  Note that there is a difference between this and start_date. Ads are usually "created" ahead of time so they'll be in the system and ready to run when the clock ticks over to midnight on the start date.
- `date_created_after=` Use this qualifier to find ads created after a given date.
- `date_created_before=` Use this qualifier to find ads created before a given date.
- `start_date=` Use this qualifier to find ads that started on or after a given date. This is useful if you'd like to give people an option to see ads added to the system in the last X days. It's also used in persistent search, to limit results to ads added since the user's last visit.
- `start_date_prior=` Use this qualifier to find ads that started before a given date.

Note that the time zone specified by the TM property attribute will be taken into account by all the date qualifiers. Also, if you query for a date too far in the past, you won't get many ads, if you get any at all. For more on this, see *The Lifecycle of a Liner*, above. There's one more date-related qualifier: date_fmt. For more on it, see *Universal Qualifiers* in "MTL Common tags."

Date & Status Defaults:

If date and status are not specified, status is set by default to active,pending. Date is not set.

If date is specified (set to today) and status is not, status is set by default to active,pending.

If date is specified (not today) and status is not, status is assumed to be active,expired,pending.

If date is specified (not today) and status is not, and **if another date-related qualifier is used**, (other than date_fmt) status ends up being active,pending.

If date and status are specified, both are used as provided.

The only statuses that will ever be set by default are active, expired and pending. If you want ads of any other status to show up, you must query explicitly for them.

The **data tags** to tell you about an ad's dates are:

- date_created - This will be the date the ad was first entered into the system. Often it does not match the start date; ads are generally uploaded from a newspaper the day before they go live.
- end_date - The last day the ad is scheduled to run
- start_date - The first day the ad is scheduled to run.

Exercises:

Ex. 1: Modify your search template to give the user a choice of search dates.

Ex. 2: Modify your detail template so it uses the search date selected by the user.

Ex. 3: Modify your detail template so that it would be possible to retrieve ads that started in the last X days. Make sure you can construct a URL to your template that takes advantage of this functionality.

Id versus Number

So far everything we've covered has focused on finding groups of ads, selected based on broad sets of criteria.

Occasionally however, you'll need to select specific ads.

There are two **qualifiers** that let you do that:

- `id=` The unique, database-assigned identifying number of an ad (think "Social Security Number.")
- `number=` The number assigned to the ad at the property. Combine this with the property qualifier to ensure uniqueness.

The difference between an ad's number and id is a local-versus-global difference.

At the property level, each ad number is (or should be) unique.

So let's say the Daily Bugle has an ad numbered 100, and one numbered 101, then 102 &etc. There is only one ad in the Daily Bugle system numbered 100. In some other city the Daily Planet has a completely different ad that also happens to be numbered 100, then a 101, 102, &etc., but there's only one ad in the Daily Planet system numbered 100.

Within the Daily Bugle's system each ad has a unique number. Same with the Daily Planet. But when both papers upload their ads to the same database those numbers conflict.

So when those ads are loaded, a new piece of information is added to each: a unique database id. The Daily Planet ad numbered 100 would have an id of maybe 2056 and the Daily Bugle's ad numbered 100 would have an id of 2057 (for instance.)

To get an ad's number or id, use these **data tags**:

- id - The ad's unique, database-assigned number.
- number - The number assigned at the newspaper.

Why keep up with an ad's newspaper-assigned number? If someone from the newspaper's classified staff needs to check an ad online, or find an ad in Ad Manipulator to kill it, that person will have no idea in the world what the database-assigned id is. But she'll have quick and easy reference to the ad number.

Exercises:

Ex. 1: Modify your detail template so that someone from your newspaper's classifieds staff could retrieve an ad by ad number.

Note, you're not assigned to modify your search template to allow search by ad number. So make sure you can construct a URL to an ad whose number you have.

Clipboard

Now let's look at the circumstances where you'd want to pull out specific, individual ads.

One nice feature of most Internet classifieds setups is the ability to let users cherry-pick ads, and then isolate what they've picked - much like circling ads in the printed newspaper, but cooler.

In the MDW system, the id's of the chosen ads are stored in a cookie. So first you'll need cookie management javascript . You can write your own or use the set of recommended scripts in Appendix B.

Now we need a way to pick an ad:

```
<A HREF="javascript:AppendCookie('ad_id','<mtl
classifieds.ad.id>')">Save</a>
```

And we need a way to get to the list once it's populated. (The cookie will be sent along automatically.):

```
<a href="http://[domain]/classifieds-
bin/classifieds?temp_type=clipboard">View My List</a>
```

If you're giving your users the option to search dates other than today, you'll want to pass their choice forward:

```
<a href="http://[domain]/classifieds-
bin/classifieds?temp_type=clipboard&date=<mcc $date>">View My List</a>
```

You probably also want to give your users the ability to clear the clipboard and start over:

```
<a href="javascript:WriteCookie('ad_id','',-20)">Clear My List</a><br>
```

For most things in MTL, it really doesn't matter what you set the sub_type of a template to or whether you link to Forge or to classifieds-bin/classified or to realestate, but for the clipboard, the user's list of cherry-picked ads, it does.

If you make a request for a clipboard template by calling classifieds-bin/classifieds or realestate, it's processed differently than if you call the Forge RPE (versus the command-line version of Forge.)

When called as "forge," the RPE automatically translates every name/value pair in a URL into variable/value pairs and passes them forward to the template. But our list of ad ids is not being passed through the URL; it's being sent via cookie. And "forge" doesn't automatically process cookies. You have to call the RPE as "classifieds-bin/classifieds" or as "realestate" to get it do to that. (If this sounds strange, think about how your own behavior changes when, for instance, your mother calls your full name in *that* voice.)

Even as classifieds-bin/classifieds and realestate, the RPE only does this when the temp_type is clipboard.

The clipboard itself can be a very simple template:

```
<mtl if value=$ad_id eq="">
     <mtl $ad_id=0>
</mtl if>

<mtl classifieds.ad id=$ad_id date=$date status=active,pending,expired>
     <mtl classifieds.ad.body><p>
</mtl classifieds.ad>
```

It only needs two qualifiers, id and either date or status. The first is easy. We qualify by id to pull out the specific ads the user identified. The if statement above the query "saves us" in case the user hits "View My List" before he puts anything on it. Without this, we'd be running a practically unqualified query - all ads, all categories, all properties. So instead we set $ad_id to 0, which brings back 0 records.

To understand why either date or status (or both) is needed, think back to the discussion about date and status, above. If you qualify by `date=today`, status is set to "`active,pending`" by default. But if your date is not "`today`," i.e. it's "`Sunday_before(tomorrow)`" or even "`today,Sunday_before(tomorrow)`" status gets set, by default, to "`active,pending,expired`."

So when we give our user a choice of dates in the search form, he's potentially looking at expired ads on the detail - and adding them to his clipboard.

If we query only by id in the clipboard, status will be set by default to "`active`," and the expired ads the user picked won't show up. So we have to either add `date=$date` to the query - which forces the default status to be active, pending *and* expired, or we have to explicitly query for ads where `status=active,pending,expired`. Or we can do both - it won't hurt anything.

The code above is the simplest version of a clipboard. You can add a little extra functionality by making sure the Appendix B javascripts get into your clipboard (if they're in your header, and your clipboard includes the header, then you're already covered) and adding this link inside the classifieds.ad query:

```
<A HREF="javascript:RemoveCookieItem('ad_id','<mtl
classifieds.ad.ad_id>')">Remove</a>
```

Of course, this just removes the ad's id from the cookie; it won't automatically erase it from the page; you'd need to refresh the browser for that.

Debug note:

- As a security feature, most browsers will not send the contents of cookies created from one domain to another domain. (E.g. Most browsers prevent someone at hackers.com

from retrieving your cookies from Amazon.com &etc.) So if you're having trouble making the ads you selected show up on your clipboard, make sure the domain your form is submitting to is the same domain that your detail is being served from.

Exercises:

Ex. 1: Create a clipboard template to display the user's cherry-picked ads.

Ex. 2: Add clipboard functionality to your detail template. Make sure you can get to your clipboard from the detail and that the right ads are displayed there.

Display Ads

While the main focus of most classifieds systems is liner ads, mdClassifieds also handles display ads. As a template writer you can choose to display those ads, in whole or in part in line with the classified liners.

The display ads in use in mdClassifieds are ads generated from mdDisplay.

mdDisplay ads have a "parent" ad and child "adlet(s)." A good example of this would be a car dealership's weekly full page ad. Typically, there are a few new car models featured at the top, then the bottom half (or so) of the ad is taken up with a grid, maybe four by five, of specific used cars on the lot: a picture, a price and some text. So all together, let's say we have 25 cars in the ad, 20 specific used cars and 5 new models. When that goes online in the classifieds, the volume of text in the ad will be too large to present together as one ad.

So it makes it more convenient to the user, and more effective for the dealer to split the ad up into logical chunks: each car becomes it's own adlet. So we have 25 adlets. Text from each of them will be extracted and inserted into the mdClassifieds database.

Any time adlets are pulled out of an ad, information about the ad they came out of is also inserted as their parent ad with type set to "display."

In the case of an ad where there are no extractable adlets, like an employment display ad, there will still be an adlet and a parent ad; it'll just be the same thing twice.

There are several display-specific **qualifiers**:

- `type=` For display ads type will be: Adlet, which is a small part of a larger ad; Display - the large, parent ad of adlets or (this type is being phased out); and PDF - a stand-alone, self-contained ad. Leave this qualifier out and by default you'll get all types. Type is null (empty) for liners.

  Because type is null for liners, you'll only want to qualify your query by type when you *don't* want the liners to show up. (Think about it… how do you query for null? If you use `type=""` then the RPE will read it as if you want all types.)

- `show_text=` Values here are "y" and "n," (yes and no.) They indicated whether the person who prepared the ad to go on line wanted the extracted text from the ad to show up with the classifieds ads. So in a template written to display liner ads, you'll generally query for `show_text=y`. Leave this out and you'll get all ads.

- `parent_id=` Pass the ad id of a display ad into this qualifier to get the display ad's child adlets.

- `subject=` Use this qualifier to query by the exact text of the ad subject. This was entered when the ad was prepared to go online. Generally, liner ads don't have subjects. The subject of an ad (display, adlet or PDF) may or may not be unique.

Notable display-specific **data tags** are:

- subject - This might be useful as a "headline" for the ad.
- type - This is rarely output to the screen, but is often used to vary presentation based on the type of the ad.
- image_link() - This tag takes an argument and returns the URL to the specified argument. Arguments are named following this pattern: [type]_image, where type is the type listed for the image in the XML output from mdDisplay.

  So, for instance, if I were trying to get the URL to the "**hirez**" image created by mdDisplay, I'd do this:

  ```
  <mtl classifieds.ad.image_link(hirez_image)>
  ```

  And I'd get back something like this:

  http://mysite.com/class_images/jpg/miltonruben6_2_03_hirez.jpg

- attribute_value() - Like image_link(), this tag takes an argument and brings back data for it. In terms of Display ads, it will generally be used to get an image's height and width:

  ```
  <mtl classifieds.ad.attribute_value(hirez_image_width)>
  ```

- parent_id - This returns the ad id of the display an adlet was extracted from. It's useful if you want to provide a link from an adlet to the full display ad.

Here's how the code might look in a detail template:

```
<mtl if {classifieds.ad.type="adlet"}>
     <mtl classifieds.ad.ad_subject><br>
     <mtl classifieds.ad.body><p>
       <a href="/classifieds-bin/classifieds?[attributes of next
       template]&parent=<mtl classifieds.ad.parent_id>&date=<mtl $date>">
       view complete ad </a>
</mtl if>
```

And the template to show the Display ad might look something like this:

```
<MAP NAME=mymap>
<mtl classifieds.ad parent_id=$parent date=$date>
       <AREA HREF="/classifieds-bin/classifieds?[attributes of next
       template]&date=<mtl $date>&id=<mtl classifieds.ad.id>"
       COORDS="<mtl classifieds.ad.attribute_value(LEFT)>,
       <mtl classifieds.ad.attribute_value(TOP)>,
       <mtl classifieds.ad.attribute_value(RIGHT)>,
       <mtl classifieds.ad.attribute_value(BOTTOM)>">
</mtl classifieds.ad>
</MAP>

<mtl classifieds.ad id=$parent date=$date>
```

```
        <b><mtl classifieds.ad.ad_subject></b><br>
        <IMG SRC="<mtl classifieds.ad.image_link(HIREZ_IMAGE)>"
        ALT="<mtl classifieds.ad.ad_subject>"
        USEMAP="#mymap" border=0
        height="<mtl classifieds.ad.attribute_value(HIREZ_IMAGE_HEIGHT)>"
        width="<mtl classifieds.ad.attribute_value(HIREZ_IMAGE_WIDTH)>">
</mtl classifieds.ad>
```

It takes two queries to do this. With the first, we get all the "children" of the parent ad and set up the image map. Then with the second, we get the parent ad itself and output the image.

Prices, Features

You should at this point have a detail template and a search template that submits to it. (If you don't, go back and do the exercises in previous chapters.)

So far you're giving your users the ability to search your classified ads by date and by keyword and the ability to pick the search logic. For some classifications of ads, more user search options are available.

Prices and some features are parsed out of ads under the Real Estate and Transportation majors (but only under these majors.)

To let your users search these ads by price, use these query **qualifiers**:

- `max_price=` This would be the high side of the user's acceptable price range. The value you pass in here needs to be numeric, i.e. no dollar signs, no commas, just numbers.
- `min_price=` This would be the low side of the user's acceptable price range. The value you pass in here needs to be numeric, i.e. no dollar signs, no commas, just numbers.

The price **data tags** are:

- price - This returns the price parsed out of the ad, e.g. $4,500.
- price_unit - Generally this value is blank, but it's meant to hold the type of currency the price is in, e.g. dollars, pesos, euros, &etc.
- raw_price - This returns the price parsed out of the ad with all the non-digit characters stripped out, e.g. 4500.

To pull out the features parsed out of Real Estate and Transportation, you use the feature **data tags**. You'll notice that each one is followed by a pair of parentheses. Into those parentheses you insert the name of the feature you want data for:

- feature_created_by() - This applies to every type of feature, although the value it returns will generally be the name of an automated loader program.
- feature_quantity() - Mileage, Beds and Baths are features whose values are stored under quantity. This tag will always return a number.
- feature_quantity_unit() - This field is generally blank, but in the case of Mileage might be miles or kilometers.
- feature_size() - Area is a feature whose value is stored under size. This tag may return words or numbers.
- feature_size_unit() - This field is generally blank, but would indicate the size unit if it were not, i.e. acres or square feet in the case of Area.
- feature_value() - Make, Model and Year are features whose values are stored under value. This tag may return words or numbers.

The features listed above are the only ones currently parsed out of classified ads. Other features may be inserted manually (via AdManipulator - contact your Support Representative for more) or automatically by a property's Ad Order Entry system.

Here's how some feature tags look "in action":

```
<mtl if {classifieds.ad.feature_quantity(BEDS)<>""}>
     <mtl classifieds.ad.feature_quantity(beds)> bedrooms
</mtl if>
```

or

```
<mtl $beds={classifieds.ad.feature_quantity(BEDS)}>
<mtl if $beds<>""}>
     <mtl $beds> bedrooms
</mtl if>
```

There are a few things to note about this example:

- Like most other MTL, the feature name is case **in**sensitive
- You can use feature data tags just like any other data tag. The fact that it has "([something here])" tacked on the end has no effect on its function.
- The second version is slightly more efficient than the first: it only makes one request for the number of bedrooms & saves the information in a variable, rather than asking for the data twice.

You may have noticed at this point that I skipped naming the **qualifiers** for features. I did that for a very good reason: there aren't any.

Classifieds.ad features are unique in MTL; it's not how you qualify your query for features (because you can't,) it's how you name the inputs in your search form.

Specifically, if you name search inputs following a certain pattern (more on this in a second), Their values will automatically be used as selection criteria when the classifieds or realestate RPE's retrieve the ads.

That means that you don't have to qualify your query by feature to narrow your results to only the ones you want. Conversely, there's no way you can prevent your search from being narrowed; it happens "automagically."

Again, this situation is unique to classifieds. Nowhere else in MTL does it matter what your inputs/variables are named.*

In fact, it's so unique in MTL that the Forge RPE doesn't do it. (That's right, if the RPE path in your URL is "forge" you can't narrow your results by features at all.) Like passing cookie values to the clipboard, features only work when you use "classifieds-bin/classifieds" or "realestate" in your URL. Up to this point, it didn't really matter which RPE your form

submitted to. But if you want to offer your users the ability to search by feature, you'll need to make sure your form action is "classifieds-bin/classifieds" or "realestate."

That said, let's look at how those inputs should be named. The general format always starts with "feature_" and follows this pattern:

```
feature_[feature name]
feature_[modifier]_[feature name]
```

It's not really important at this point that you know this, but be aware that features are stored apart from the classified ads. They're in a separate table where every feature has a spot for quantity, size, and value. Although it is generally not the case, each feature could have data in each of those spots. For convenience, I'll refer here to features by the type of data they *normally* have, but be aware that this is *only* a convenience.

For feature values, you can search for specific values:

```
feature_[feature name]
```

Or you can search for value ranges

```
feature_start_[feature name]
feature_end_[feature name]
```

Or you can enter a comma delimited list of values and retrieve any record with one of them:

```
feature_includes_[feature name]
```

Or you can enter a string and find records where the feature value contains it. (E.g. you enter "hon" to find Hondas.)

```
feature_contains_[feature name]
```

Translated into actual features, it might look like this in a form:

```
Make: <input type=text name="feature_make"><br>
Model: <input type=text name="feature_model"><p>

Year from <input type=text name="feature_start_year">
to<input type=text name="feature_end_year"><br>
<div align=center>or</div><br>
Enter the specific years you're looking for, separated by commas:
<input type=text name="feature_includes_year">
```

And like this in a URL:

```
…&feature_make=honda&feature_model=accura&feature_end_year=&feature_start
_year=98&feature_includes_year=…
```

In the URL snippet above, I've filled in values for almost everything – except the end of the year range. So I'll get the default: 0. If I used a start value but no end value, I'd get that default: 2999.

Like year, make and model are "values," so it's possible to let users search them by range. But it's probably not advisable. Let's say you let your users search Make by range, for instance:

```
Make from <input type=text name="feature_start_make">
to<input type=text name="feature_end_make">
```

If I fill in "Honda" as my start, but leave the end of the range blank, just like with year, the end of the range is set to 2999. So I'll end up searching for makes from, "Honda" to 2999. But since 2999 comes before Honda in an ascii-betical sort, I'll get nothing back - even though the database may be full of Hondas, Lincolns, Toyotas, etc.

Normally, you'd be able to handle this in your template with an if statement before the query by testing for a blank value & "correcting" it by dropping in something like "zzzz" for the end of the range. But again, the feature search criteria get set automagically before the RPE even gets to your template, so if you're interested in offering this functionality, you'll have to test for and correct a blank end-range value at the browser using JavaScript.

For feature quantities, you can search only by range:

```
feature_min_[feature name]
feature_max_[feature name]
```

Translated to actual features, it might look like this in a form:

```
Beds from <input type=text name="feature_min_beds">
to <input type=text name="feature_max_beds"><br>
```

And like this in a URL:

```
&feature_min_beds=3&feature_max_beds=5
```

You cannot search feature quantities for specific values except by using the same number in both the min and the max.

Note the syntax difference between value range (start/end) and quantity range (max/min.)

For feature sizes, you can search only for specific values. This is another search option that is possible, but not necessarily practical:

```
feature_[feature name]
```

Translated to actual features, it might look like this in a form:

```
Square footage <input type=text name="feature_area">
```

This would bring back only ads that matched the exact value you entered. So if you search for feature_area=2100, you'll only get ads with that exact value, and not an ad with an area of 2099 or one with an area of 2110.

For all feature types, you've got one more options:

`feature_exists_[feature name]` - Simply verifies that a feature with the given name exists.

For another approach to features, see *Classifieds.Ad_Feature* below

Exercises:

Copy your search and detail templates to your Transportation classification and your Real Estate classification. (TM's "Import Template(s)" function works beautifully for this.) You should now have three identical copies of each template.

Ex. 1: Add inputs to your Transportation search form to allow the user to specify the make and model and ranges for year, price and mileage. Be sure to change the form action to use "classifieds-bin/classifieds."

Ex. 2: Modify your Transportation detail template to honor the price range specified in the search (remember, features will be honored automagically.)

Add data tags to your Transportation detail to print out the make, model, year, price and mileage.

Ex. 3: Add inputs to your Real Estate search form, to allow the user to specify the number of bedrooms, bathrooms and price range. Be sure to change the form action to use "classifieds-bin/classifieds."

Ex. 4: Modify your Real Estate detail template to honor the price range specified in the search.

Add data tags to your Real Estate detail to print out the number of beds and baths, the area and the price.

* Okay, it's *almost* unique in MTL. As I stated in *Classifieds.Ad - basic query*, if you've got page=3 in your URL you'll automagically get page three of your ads. But unlike the feature behavior, the pagination behavior *can* be overridden.

Ordering

We've touched briefly on ordering your results in the Keywords section; if a user performs a keyword search, it make sense to bring the most relevant results to the top of the list.

The general MTL order syntax is this:

```
orderby=colname,colname,colname,&etc.
```

This applies whether you want to order by the values in an actual column, such as body, or by calculation values, like search_score. (For more on ordering, see *Sort of…* in "MTL Basic Concepts.")

In ads with features, it also makes sense to order by those features. To orderby features, the syntax is pretty much the same. In our real estate detail we might order this way:

```
orderby=search_score,feature_quantity(beds),feature_quantity(baths)
```

Notice that the values passed in to this orderby tag still start with search_score. If the user didn't enter keywords, all ads' search_scores will be equal: 0. So then they'll all be sub-ordered by the next value: feature_quantity(beds) as if search_score didn't even show up in the orderby list.

You might also consider ordering by price, descending (to put the cheapest first) or descending by start_date to pull up the newest ads first.

On the other hand, you may want to let the user pick the ordering criteria. See *Appendix C: User-defined ordering* for an example.

Ordering Notes:

- While the number of orderby values you can use is virtually unlimited, scattered in that unlimited number of values, you can only list a total of four features.
- Advanced sorting i.e. `orderby=colname(val1,val2)`, explained in "MTL Basic Concepts" does not work for the classifieds.ad query. It was not implemented because the result sets for classifieds.ad are usually very large, and the larger the result set, the more CPU-intensive that particular function becomes.

Exercises:

Ex. 1: Implement feature ordering in your real estate detail template.

Ex. 2: Implement feature ordering in your transportation detail template.

Classifieds.Ad_Feature

If you know the names and types of an ad's features, you can display the values from within your classifieds.ad query. But if you're not sure how they're named, or if you don't want to have to explicitly name all of them, you'll use this query.

Here's a brief rundown of some **qualifiers**:

- `ad_id=` Pass an ad's id, or multiple, comma-delimited id's here to get features for a specific ad.
- `feature_id=` Pass in the unique id of a specific feature to retrieve that row.
- `max_quantity=` The maximum value you want in a quantity field. Similar to the `"feature_max_[feature name]"` search input
- `min_quantity=` The minimum value you want in a quantity field. Similar to the `"feature_min_[feature name]"` search input
- `quantity=` Unlike in classifieds.ad, you can search classifieds.ad_feature for an exact quantity. Use this qualifier to do so.
- `quantity_unit=` The quantity unit field is often blank, but use this qualifier to find records where quantity is measured in a given unit.
- `size=` As with the size search input, use this to find feature records with a certain, exact size value.
- `size_unit=` The size unit field is often blank, but use this qualifier to find records where size is measured in a given unit.
- `type=` Type corresponds to the feature name i.e. transportation records might have features of the "make" type or "model" or "mileage" type.
- `value=` Use this qualifier to find records with a given value in the value field.

Although it is generally not the case, each feature row may have data in the quantity column, the size column and the value column.

Here's a quick and dirty way to run through all an ad's features with the classifieds.ad_feature query:

```
<mtl classifieds.ad_feature ad_id=classifieds.ad.id>
     <mtl classifieds.ad_feature.type case=title>:
     <mtl classifieds.ad_feature.value>
     <mtl classifieds.ad_feature.size>
     <mtl classifieds.ad_feature.size_unit>
     <mtl classifieds.ad_feature.quantity>
     <mtl classifieds.ad_feature.quantity_unit>
     <br>
</mtl classifieds.ad_feature>
```

We can safely use most of the data tags here because the ones that don't apply won't show up in our output. Here's sample Transportation output from this query:

Make: TOYOTA
Model: CAMRY
Year: 1987

Real Estate output might look like this:

Baths: 2
Beds: 3
Garage: 2

Note that we qualified our classifieds.ad_feature query using a classifieds.ad data tag. Of course this will only work inside a classifieds.ad query. So our structure actually looks more like this:

```
<mtl classifieds.ad property=$property classification=$classification
[other qualifiers]>
     <mtl classifieds.ad_feature ad_id=classifieds.ad.id>
          <mtl classifieds.ad_feature.type case=title>
          [other data tags]<br>
     </mtl classifieds.ad_feature>
     <mtl classifieds.ad.body><p>
<mtl classifieds.ad>
```

For classified ads from a newspaper's front-end system, with a limited number of features, this may or may not be the best way to go. But for MLS records, for instance, where you have lots of feature data, this might be the best way to pull the information out.

Customers and Real Estate

mdClassifieds was originally built to handle the ads from newspaper front-end systems, but what has evolved is a supremely flexible data structure that also handles other types of information very well.

A case in point is MLS listings. Each home has a price, a description (body), a listing id (number,) features, etc.

Homes for sale, however, generally also have a listing agent. Classified ads don't. But a classified ad does have a customer (this data is often not transmitted from the newspaper to the mdClassifieds database, but there are fields for the data anyway.)

So in this case, the listing agent becomes the customer for the ad. And the customer info you can retrieve with the ad (name, phone, address, &etc.) is the agent's data.

You can query for all an agent's homes using these **qualifiers**:

- `customer_id=` This is a unique, database-assigned id. Use it to get all of a customer's ads.
- `agent_id=` This is the id that MLS assigned to the realtor. It must be combined with property to ensure uniqueness. Use it to get all of an agent's ads.

Each ad has both fields: customer_id and agent_id. The numbers will be different, but they'll both always represents the same person: the real estate agent who listed the home. (Non-real estate ads can also have customers but *those* customers may or may not have agent id's.)

If you qualify a query by an agent's customer_id, you'll get all 48 of that agent's listings. And if you qualify a query by an agent's agent_id and property, you'll get the same 48 listings. So why two numbers? Why not just whittle it down to one and be done with it?

Well, customer_id is required by the database. It gets assigned automatically as new customers (agents) are entered. It is a totally unique id, and no other customer in mdClassifieds has ever had that id or ever will.

Agent_id, on the other hand, may or may not be unique in the database. There may be an agent #48 in your town, and another in the town down the road. So you can't rely on the agent id for uniqueness. However. Every agent will know his agent id by heart (and have no clue what his customer number is.)

Also, while every agent is a customer, not every customer is an agent.

So, we've established that ads have customers (agents.) There's one more level in the relationship. Agent's have brokers.

And just like you want to be able to retrieve all of an agent's ads, you might also want to pull back the ads of all the agents under a given broker.

E.g. Let's say Tom Broker wants to pay you extra to promote all the homes that are listed under him (i.e. his listings and his agents' listings.) You could do it by getting a list of his

agents, and hard-coding their ids into your template - and hoping that he doesn't gain or lose any agents.

Or you could **qualify** by:

- `broker_id=` The id number MLS assigned a broker. This must be combined with property to ensure uniqueness.

The same way customer_id and agent_id are stored with each ad, broker_id is stored with each ad too. (Non real estate ads have this field too, but it's generally blank.)

Like agent_id, broker_id is only unique within a property. So if you're qualifying by broker_id you *must* also qualify by property - otherwise you'll get your area's broker #4 **and** all the other areas' broker #4s.

Be aware that brokers also have customer ids. Broker information is stored in the same place, with the same fields as agent information. But you **can't** get the listings under a broker using the broker's customer id - because the broker isn't the ad's customer. The agent is.

Ad_Customer, Customer_Attribute

Each classified ad has a customer. Data about those customers is not usually transmitted from newspapers' front-end systems, but sometimes information about a customer is available in the database - particularly if the ad has been entered from the Web, or if the ad is from an MLS feed.

You can get most of ad customer's information from within the classifieds.ad query. Or, if you want something like a list of real estate agents, for example, you can skip the ads and go straight to the customer with the classifieds.ad_customer query.

Use this query to retrieve information about a single user or list of users, or to find users that meet specific criteria. (Such as first_name=John.)

Classifieds.ad_customer q**ualifiers** of note are:

- `agent_id=` Return customers with this agent ID. This is different from customer id, and must be combined with the property qualifier for uniqueness.
- `is_broker=` Set this to 'Y', to return only brokers; 'N' to exclude brokers.
- `broker_id=` Return customers with the specified broker ID - again different from customer_id. This qualifier must be combined with the property qualifier for uniqueness. Use this qualifier to pull up the specified broker *and* all his/her agents.
- `parent_id=` Return customer records with the specified parent record ID. If you use the *customer id* (not broker id - They're different) of a broker, you'll get all that broker's agents, *but not the broker* - because he's not his own parent. Use an agent's customer id and you'll likely get nothing back - because agent records don't have child records.
- `attribute_contains()=` Usage takes this form: `attribute_contains(`*`arg`*`)=`*`val`* Return customers that have an attribute named *arg* the value of which contains the specified *val*.
- `attribute_exists()=` Return customers for whom this attribute exists.

  Usage example: `attribute_exists(BROKER_ID)=NO` means that the customer does not have BROKER_ID as an attribute. Possible values are 'Yes', 'True', 'On' or 'No', 'False' and 'Off'.
- `attribute_value(val)=` Usage takes this form: `attribute_value(`*`arg`*`)=`*`value`* Return customers that have an attribute named *arg* of the specified *value*. (i.e. `attribute_value(BROKER_ID)=573375`)

Note that most of the qualifiers listed here are not things you can order the results by.

Classifieds.ad_customer **Data tags** of note are:

- attribute() - Get the value of one of the customer's specified attribute.

- parent_attribute() - Get the value of the specified attribute from the customer's parent record e.g. get an attribute of an agent's broker.
- parent_id - Get the customer id of the customer's parent record (if there is one.) E.g. this tag would give you the customer id of an agent's broker.
- password - Customer's password. This is generally blank.
- type - Customer type. This is generally blank.

So, you can get to most of a customer's information from classifieds.ad, and you can get to all of it from classifieds.ad_customer - if you know what attribute names the customer has. But if you don't know the attribute names - or if you're lazy, you might want to use the classifieds.customer_attribute query to pull by a customer's attributes. Use this query to run through each attribute one by one.

Classifieds.customer_attribute **qualifiers** of note are:

- `customer_id=` This is the id of the customer whose attributes you're looking for.
- `key=` This is the name of the attribute.
- `type=` Return only records for attributes of this type; there may be one or more types.
- `value=` Return attributes with the specified value.

The **data tags** for classifieds.customer_attribute are pretty straightforward, and won't be covered here in depth.

Customer attributes are usually extra pieces of data that there's just not a spot for in the customer table. For instance, there are columns in the customer table for day phone and night phone, but not for fax number or pager number. So those are things that would get stored as customer_attribute(s) - probably with keys named something like "fax" and "pager."

As an example, here's how to get a list of customers given a broker id:

```
<mtl classifieds.ad_customer broker_id=$broker
orderby=parent_id,last_name,first_name>
   <mtl if {classifieds.ad_customer.parent_id="" and
   (classifieds.ad_customer.first_name<>"" or
   classifieds.ad_customer.last_name<>"")}>
        <b><mtl classifieds.ad_customer.first_name>
        <mtl classifieds.ad_customer.last_name> and his team of agents
        are eager to help you find your next home!</b><p>
   <mtl else>
        <mtl classifieds.ad_customer.first_name>
        <mtl classifieds.ad_customer.last_name><br>
     <mtl if {classifieds.ad_customer.day_phone <> ""}>
               Days: <mtl classifieds.ad_customer.day_phone><br>
  </mtl if>
     <mtl if {classifieds.ad_customer.night_phone <> ""}>
      Nights: <mtl classifieds.ad_customer.night_phone><br>
```

```
      </mtl if>
      <!mtl "--- &etc… ---">
         <mtl classifieds.customer_attribute
         customer_id=classifieds.ad_customer.id>
            <mtl classifieds.customer_attribute.key case=title> :
            <mtl classifieds.customer_attribute.value case=title><br>
            </mtl classifieds.customer_attribute>
    </mtl if>
      <p>
</mtl classifieds.ad_customer>
```

There are a couple things to note about this example:

- We order first by parent_id to bring the broker to the top - he won't have a parent. We can also use that to test for his record and output the welcoming message - if the broker's name is available.
- Because we can't order by broker_id, the ordering scheme breaks down if $broker has more than one id in it. (Try it yourself and see.)
- Actually, broker_id is an attribute, so it would work to use multiple broker_id's in $broker if we could order the ad_customer query by attribute(broker_id), but attribute's not a valid orderby option.
- Our template pulls out all attributes, but practically speaking, there will be ones you don't want displayed. Before you put something like this into production, you'll want to see what a basic customer_attribute query like this returns, then qualify your query to narrow down the results to just want you want.

  In doing this, keep in mind that it's always better to filter what you get back by adding qualifiers to your query - and retrieving fewer results - rather than using if statements to suppress output of records you don't want.

Classifieds.Attachments

Attachments in classifieds are generally URL's - addresses to logos, pictures, VR tours &etc., - but could be any other type of text.

Attachments aren't "attached" to any one type of record. Any kind of record (object) could have an attachment - but the objects most likely to have attachments are ads and customers.

To find attachments you'll use the classifieds.attachments query. Here are some **qualifiers** of note:

- `attachment_id=` This is the unique id, or list of id's of an individual attachment.
- `attachment_type=` Return attachments of the specified type, i.e. 'URL', 'HYPERTEXT', etc.
- `name=` Return attachments with the specified name.
- `object_id=` Return attachments belonging to the specified ID, normally an ad id or ad_customer id. This qualifier must be combined with object_type for uniqueness.
- `object_type=` Return attachments belonging to objects of the specified type. Valid values for this are classified_ad and ad_customer.

Usually, you can take the unique id from one area, like an ad id, and use it to query another area, like ad_feature, and be sure you're pulling back *only* the records that you were looking for - i.e. only the features that belong to the ad whose id you used.

But because attachments can be attached to any type of object - an ad, a customer, even another attachment, it's a little different in classifieds.attachments.

Just like ad numbers are unique at the properties where the ads are generated, but not unique within the database (see *Id versus Number*, above) ad id's are unique within the classifieds.ad table - there will never be another ad in the classifieds.ad table with that id - and customer id's are unique within the classifieds.ad_customer table. But when you look at both of those tables together - which is what classifieds.attachments does, you may find that there are two records with the same id.

So to make sure you're getting the attachments only for the record you're looking for, you must combine `object_id=` with `object_type=` in your query.

Attachment **data tags** of note are:

- text_attachment - This is the actual "attachment." For attachment types 'URL' and 'HYPERTEXT', this should be a URL.
- attachment_id - Unique id of attachment.
- attachment_type - Type of attachment, i.e. 'URL', 'HYPERTEXT', etc.
- description - Text description of attachment.
- name - Name of attachment. This is often something like "Agent Photo."

- object_id - The id of the object the attachment belongs to.
- object_type - This is the type of object the attachment belongs to.

Here's a sample use:

```
<mtl classifieds.attachments
object_id=classifieds.ad_customer.customer_id object_type=AD_CUSTOMER
attachment_type=HYPERTEXT NAME=AGENT_PHOTO no_data=continue>
  <mtl if val=classifieds.attachments.text_attachment ne="">
       <img src="<mtl classifieds.attachments.text_attachment>" width=70
       height=100>
  <mtl else>
    <img src="http://[domain2]/images2/blank.gif" width=70 height=100>
    </mtl if>
</mtl classifieds.attachments>
```

Classifieds.Object_Attribute

Like customer_attributes, object_attributes are extra pieces of information - usually about ads or attachments.

For instance, mdDisplay ad attributes such as image sizes and coordinates are stored here. Generally this data is accessed through the classifieds.ad tag set, but the classifieds.object_attribute query is available if you want direct access.

For the same reasons discussed in *Classifieds.Attachments*, above, you'll need to combine object_id and object_type for uniqueness.

Query for object_attributes with these **qualifiers**:

- `object_attribute_id=` Unique id of an object_attribute .
- `attribute_name=` The name of the attribute you're looking for.
- `attribute_value=` Returns records with the specified attribute value.
- `object_id=` The id of the object the attribute belongs to.
- `object_type=` This is the type of object the attribute belongs to.

The **data tags** for classifieds.customer_attribute are pretty straightforward, and won't be covered here in depth.

Classifieds.Codesets

The codesets area of classifieds is what's called a "look up table." It holds the kind of values you populate form options with, and that's how it's most often used.

The codesets table has a fairly narrow focus, and therefore just a few query **qualifiers**:

- `category=` Return codesets records in the specified category.
- `id=` Get records with the specified ID or comma-delimited id's.
- `key=` Get records with the specified keys.
- `value=` Get records with the specified values.

Here's a possible use

```
<select name="pool options">
<mtl classifieds.codesets category='MLS POOL'>
     <option value="<mtl classifieds.codesets.value>"><mtl
classifieds.codesets.value></option>
</mtl classifieds.codesets>
</select>
```

And here's the output:

```
<select name="pool options">
      <option value="Pool">Pool</option>
      <option value="In-Ground Pool">In-Ground Pool</option>
      <option value="Above Ground Pool">Above Ground Pool</option>
      <option value="Hot Tub, Spa, Jacuzzi">Hot Tub, Spa, Jacuzzi</option>
      <option value="Pool and Hot Tub/Spa/Jacuzzi">Pool and Hot Tub/Spa/Jacuzzi</option>
</select>
```

Codesets values are also used to populate most of the dropdowns in MDW's Java clients. For instance, TM's list of template types (calendar, classifieds, dynapub, &etc.) and sub-types comes out of the codesets table.

Appendix A: Exercise Answers

Accessing a template

Ex. 1: For help with this, contact your Support representative.

Classifieds.Category

Ex. 1: Consult the examples in the chapter.

Ex. 2:

```
<mtl classifieds.category property=$property
classification=$classification show_children=3>
     <mtl if {classifieds.category.number=''}>
          <b><mtl classifieds.category.name case=title>:</b><br>
     <mtl else>
            <mtl classifieds.category.number>
          <mtl classifieds.category.name case=title><br>
     </mtl if>
</mtl classifieds.category>
```

Ex. 3:

```
<mtl if {$classification=''}>
     <mtl $classification=[your default major]>
</mtl if>

<form action="/classifieds-bin/classifieds" method="get">

<select name="categorynumber">
<mtl classifieds.category property=$property
classification=$classification>
     <option value="<mtl classifieds.category.number>"><mtl
classifieds.category.name case=title></option>
</mtl classifieds.category>
</select>
<input type=submit value=" Go ">
</form>
```

Classifieds.Ad - basic query

Ex. 1

```
<mtl if {$classification=''}>
     <mtl $classification="Real Estate for Sale">
</mtl if>
<mtl if {$adsperpage=''}>
```

```
     <mtl $adsperpage=25>
</mtl if>

<font size=+1><b><mtl $classification case=title>:</b></font><p>

<mtl classifieds.ad property=$property classification=$classification
          category_number=$categorynumber ads_per_page=$perpage
page=$page>
<mtl if {classifieds.ad.index=1}>
     <div align=center>
     Displaying <mtl classifieds.ad.range_min> - <mtl
classifieds.ad.range_max>
     of <mtl classifieds.ad.adcount> ads found.<br>
     <mtl classifieds.ad.page_nav_bar></div><p>
</mtl if>
     <mtl classifieds.ad.body><p>
</mtl classifieds.ad>
```

Ex. 2: Code copied from Ex. 3 above. New code in italics. The input for tl may not be necessary for your template set.

```
<mtl if {$classification=''}>
     <mtl $classification=[your default major]>
</mtl if>

<form action="/classifieds-bin/classifieds" method="get">

     <input type=hidden name=temp_type value=detail>
     <input type=hidden name=tl value=[your tl]>
     <input type=hidden name=classification value="<mtl
$classification>">

     <select name="categorynumber">
     <mtl classifieds.category property=$property
classification=$classification>
          <option value="<mtl classifieds.category.number>"><mtl
classifieds.category.name case=title></option>
     </mtl classifieds.category>
     </select><br>

     Ads per page:
     <select name=perpage>
          <option value=10>10</option>
          <option value=15>15</option>
          <option value=20>20</option>
          <option value=25>25</option>
     </select><br>

     <input type=submit value=" Go ">
</form>
```

Date, Status

Ex. 1: Only new inputs shown

```
<input type=radio name=date value="today" selected>Today
<input type=radio name=date value="sunday_before(today)">Sunday
<input type=radio name=date value="today,sunday_before(today)">Both<br>
```

Ex. 2: Code copied from Ex. 1 above. New code in italics. Some code (indicated by ellipses) omitted for brevity.

```
…
<mtl if {$date=''}>
     <mtl $date=today>
</mtl if>

…

<mtl classifieds.ad date=$date property=$property
classification=$classification category_number=$categorynumber
ads_per_page=$perpage page=$page>
     …
     <mtl classifieds.ad.body><p>
</mtl classifieds.ad>
```

Ex. 3:

Part A: Add start_date=$sdate (or whatever you'd like your variable named) to your query.

Part B: Construct a URL to your template. Then add &sdate=today-3 or &sdate=today-5 &etc.

Id versus Number

Ex. 1:

```
…
<mtl classifieds.ad number=$adnumber [other qualifiers]>
…
```

You've got two options for letting your classifieds staff pull up ads with the new functionality. Tell them to add &adnumber=X to a classifieds URL, or build them a (non-public) search form.

Keywords

Ex. 1: Some code (indicated by ellipses) omitted.

```
…
<mtl classifieds.ad keyword_font="<font color='#cc3333'><b>,</b></font>"
keyword_query=$keyword keyword_search_logic=$logic [other qualifiers]
orderby=search_score>
```

```
     …
     <mtl classifieds.ad.body><p>
</mtl classifieds.ad>
```

Ex. 2: Only new inputs are listed below.

```
Keyword: <input type=text name="keyword"><p>
Search for records with
<input type="radio" name=logic value="and" selected>All search terms<br>
<input type="radio" name=logic value="or">At least one term<br>
<input type="radio" name=logic value="fuzzy">Similarly spelled words<br>
<input type="radio" name=logic value="about">Related concepts<br>
<input type="radio" name=logic value="soundex">Words that sound like this
<input type="radio" name=logic value="stemming">Words with this stem
("read" expands to "reading," "reader," etc.)
```

Clipboard

Ex. 1:

```
[Insert javascript from Appendix B]

<mtl if value=$ad_id eq="">
     <mtl $ad_id=0>
</mtl if>

<a href="/classifieds-bin/classifieds?temp_type=clipboard&tl=[your
tl]&date=<mtl $date>">Refresh My List</a>


<mtl classifieds.ad id=$ad_id date=$date status=active,expired>
     <mtl classifieds.ad.body><br>
     <A HREF="javascript:RemoveCookieItem('ad_id','<mtl
classifieds.ad.ad_id>')">Remove</a>
     <p>
</mtl classifieds.ad>
```

Ex. 2: Code copied from above. Some code (indicated by ellipses) omitted for brevity. New code in italics.

```
…

<mtl classifieds.ad [qualifiers] orderby=search_score>

<mtl if {classifieds.ad.index=1}>
     <div align=center>
     Displaying
     <mtl classifieds.ad.range_min> -
     <mtl classifieds.ad.range_max>
```

```
     of <mtl classifieds.ad.adcount> ads found.<br>
     <mtl classifieds.ad.page_nav_bar><br>

  <a href="<mcc classifieds.ad.nav_url>?temp_type=clipboard&tl=[your
  tl]&date=<mtl $date>">View My List</a>
  <a href="javascript:WriteCookie('ad_id','',-20)">Clear My
  List</a></div><p>
</mtl if>
     <mtl classifieds.ad.body><br>
     <A HREF="javascript:AppendCookie('ad_id','<mtl
classifieds.ad.id>')">Save</a><p>
</mtl classifieds.ad>
```

Prices, Features

Ex. 1: Only new inputs are listed below. Dropdowns are used for price to keep the user from entering non-numeric characters.

```
Make: <input type="text" name="feature_make"><br>
Model" <input type="text" name="feature_model"><br>

Year: <input type="text" name="feature_start_year">
to <input type="text" name="feature_end_year"><br>

Mileage:
 <select name="feature_min_mileage">
     <option value=""></option>
<mtl loop times=15>
     <option value="<mtl {loop.index*10000}>"
     <mtl if {$feature_max_mileage=loop.index*10000}> selected</mtl if>>
     <mtl {loop.index}>0,000</option>
</mtl loop>
</select>
to
<select name="feature_max_mileage">
     <option value=""></option>
<mtl loop times=15>
     <option value="<mtl {loop.index*10000}>"
     <mtl if {$feature_max_mileage=loop.index*10000}> selected</mtl if>>
     <mtl {loop.index}>0,000</option>
</mtl loop>
</select><br>

Price:
<select name="adpricemin">
     <option value = ""> --- </option>
<mtl loop times=10>
     <option value="<mtl {loop.index*500}>"
     <mtl if {$adpricemin=loop.index*500}> selected</mtl if>>
     $<mtl {loop.index*5}>00</option>
</mtl loop>
</select>
to
<select name="adpricemax">
     <option value = ""> --- </option>
```

```
<mtl loop times=10>
     <option value="<mtl {(loop.index*loop.index)*1000}>"
     <mtl if {$adpricemin=(loop.index*loop.index)*1000}> selected</mtl
if>>
     $<mtl {(loop.index*loop.index)}>,000</option>
</mtl loop>
</select>
```

Ex. 2: Template copied from above. Some code (indicated by ellipses) omitted for brevity. New code in italics.

```
…
<mtl classifieds.ad [other qualifiers] min_price=$adpricemin
max_price=$adpricemax orderby=search_score>
     …
     <mtl if {classifieds.ad.price<>""}>
          <mtl classifieds.ad.price><br>
     </mtl if>
     <mtl if {classifieds.ad.feature_value(year) <> ""}>
          <mtl classifieds.ad.feature_value(year)><br>
     </mtl if>
     <mtl if {classifieds.ad.feature_value(make) <> ""}>
          <mtl classifieds.ad.feature_value(make)><br>
     </mtl if>
     <mtl if {classifieds.ad.feature_value(model) <> ""}>
          <mtl classifeids.ad.feature_value(model)><br>
     </mtl if>
     <mtl if {classifieds.ad.feature_quantity(mileage) <> ""}>
          <mtl classifieds.ad.feature_quantity(mileage)><br>
     </mtl if>

     <mtl classifieds.ad.body><p>
</mtl classifieds.ad>
```

Ex. 3: Only new inputs are listed below. The price selects are output conditionally because renters will be looking for a different price range than buyers.

```
Bedrooms: <input type="text" name="feature_min_beds">
to <input type="text" name="feature_max_beds"><br>

Bathrooms: <input type="text" name="feature_min_baths">
to <input type="text" name="feature_max_baths"><br>

Price:
<select name="adpricemin">
     <option value = "">---</option>
<mtl if {$classification="real estate for sale"}>
     <mtl loop times=13>
          <option value="<mtl {(loop.index+2)*20000}>"
          <mtl if {$adpricemin=(loop.index+2)*20000}> selected</mtl if>>
          $<mtl {(loop.index+2)*20}>,000</option>
     </mtl loop>
```

```
<mtl else>
     <mtl loop times=13>
          <option value="<mtl {loop.index*100}>"
          <mtl if {$adpricemin=loop.index*100}> selected</mtl if>>
          $<mtl {loop.index*100}></option>
     </mtl loop>
</mtl if>
</select>
to
<select name="adpricemax">
     <option value = "">---</option>
     <mtl if {$classification="real estate for sale"}>
          <mtl loop times=13>
               <option value="<mtl {(loop.index+2)*30000}>"
               <mtl if {$adpricemax=(loop.index+2)*30000}> selected</mtl
if>>
               $<mtl {(loop.index+2)*30}>,000</option>
          </mtl loop>
     <mtl else>
          <mtl loop times=13>
               <option value="<mtl {(loop.index*100)+200}>"
               <mtl if {$adpricemin=(loop.index*100)+200}> selected</mtl
if>>
               $<mtl {(loop.index*100)+200}></option>
          </mtl loop>
     </mtl if>
</select>
```

Ex. 4: Query copied from above. Some code (indicated by ellipses) omitted for brevity. New code in italics.

```
…
<mtl classifieds.ad [other qualifiers] min_price=$adpricemin
max_price=$adpricemax orderby=search_score>
     …
     <mtl if {classifieds.ad.price<>""}><mtl
classifieds.ad.price><br></mtl if>
     <mtl if {classifieds.ad.feature_quantity(beds) <> ""}>
          <mtl classifieds.ad.feature_quantity(beds)>bedrooms<br>
     </mtl if>
     <mtl if {classifieds.ad.feature_quantity(baths) <> ""}>
          <mtl classifieds.ad.feature_quantity(baths)>bathrooms<br>
     </mtl if>
     <mtl if {classifieds.ad.feature_size(area) <> ""}>
          <mtl classifieds.ad.feature_size(area)><br>
     </mtl if>

     <mtl classifieds.ad.body><p>
</mtl classifieds.ad>
```

Ordering

Ex. 1: Simple example below. For a more elaborate example, see Appendix C

```
<mtl classifieds.ad [other qualifiers]
orderby=search_score,price,feature_quantity(beds):d,feature_quantity(bath
s):d>
```

Ex. 2: Simple example below. For a more elaborate example, see Appendix C

```
<mtl classifieds.ad [other qualifiers]
orderby=search_score:price,feature_value(year):d,feature_quantity(mileage
):d,feature_value(make),feature_value(model)>
```

Appendix B: Cookie Javascripts

```
<script language="JavaScript">
<!--
function WriteCookie (cookieName, cookieValue, expiry) {
     var expDate = new Date();
     if(expiry)     {
          expDate.setTime (expDate.getTime() + expiry);
document.cookie= cookieName + "=" + escape(cookieValue) + "; expires=" +
expDate.toGMTString();
     } else {
          document.cookie = cookieName + "=" + escape(cookieValue);
     }
}

function setCookie (CookieName, CookieValue) {
     WriteCookie(CookieName, CookieValue, 0);
}

function ReadCookie (CookieName) {
     var CookieString = document.cookie;
     var CookieSet = CookieString.split (';');
     var SetSize = CookieSet.length;
     var CookiePieces
     var ReturnValue = "";
     var ReturnValue2 = "";
     var x = 0;

     for (x = 0; ((x < SetSize) && (ReturnValue == "")); x++) {
          CookiePieces = CookieSet[x].split ('=');

          if (CookiePieces[0].substring (0,1) == ' ') {
               CookiePieces[0] = CookiePieces[0].substring (1,
CookiePieces[0].length);
          }
     }
}

function AppendCookie (CookieName, itemToAppend) {
     var CookieString = document.cookie;
     var CookieSet = CookieString.split (';');
     var SetSize = CookieSet.length;
     var CookiePieces
     var CookieValue = "";
     var CookieFound = 0;

     var x = 0;
     for (x = 0; ((x < SetSize) && (CookieValue == "")); x++) {
          CookiePieces = CookieSet[x].split ('=');
          if (CookiePieces[0].substring (0,1) == ' ') {
               CookiePieces[0] = CookiePieces[0].substring (1,
CookiePieces[0].length);
          }

          if (CookiePieces[0] == CookieName) {
               CookieFound = 1;
```

```
               CookieValue = unescape(CookiePieces[1]) + ',' +
itemToAppend;
               WriteCookie(CookiePieces[0], CookieValue, 0);
          }
     }
     if (CookieFound == 0) { WriteCookie(CookieName, itemToAppend, 0); }
}

function RemoveCookieItem (CookieName, itemToRemove) {
     var CookieString = document.cookie;
     var CookieSet = CookieString.split (';');
     var SetSize = CookieSet.length;
     var CookiePieces
     var CookieSubPieces
     var CookieValue = "";
     var SubSetSize;

     var x = 0; var i = 0;

     for (x = 0; ((x < SetSize) && (CookieValue == "")); x++) {
          CookiePieces = CookieSet[x].split ('=');

          if (CookiePieces[0].substring (0,1) == ' ') {
               CookiePieces[0] = CookiePieces[0].substring (1,
CookiePieces[0].length);
          }

          if (CookiePieces[0] == CookieName) {
               CookieSubPieces = unescape(CookiePieces[1]).split(',');
               SubSetSize = CookieSubPieces.length;
               for (i = 0; i < SubSetSize; i++) {
                    if (CookieSubPieces[i] != itemToRemove)     {
                         if (i==0) {
                              CookieValue = CookieSubPieces[i];
                         }else{
                              CookieValue = CookieValue + ',' +
CookieSubPieces[i];
                         }
                    }
               }
               if (CookieValue == ',')  { CookieValue = ''; }
               WriteCookie(CookiePieces[0], CookieValue, 0);
          }
     }
}

function ValidateCookie(CookieName)  {
     var CookieString = document.cookie;
     var CookieSet = CookieString.split (';');
     var SetSize = CookieSet.length;
     var CookiePieces
     var CookieValue = "";
     var CookieFound = 0;
     var x = 0;

     for (x = 0; ((x < SetSize) && (CookieValue == "")); x++) {
          CookiePieces = CookieSet[x].split ('=');
```

```
          if (CookiePieces[0].substring (0,1) == ' ') {
               CookiePieces[0] = CookiePieces[0].substring (1,
CookiePieces[0].length);
          }
          if (CookiePieces[0] == CookieName) {
               CookieFound = 1;
               return;
          }
     }

     if (CookieFound == 0) { WriteCookie(CookieName, '1', 0); }
}
//-->
</script>
```

Appendix C: User-defined ordering

This example, from a real estate template, is easily adapted to other feature sets. Some code (indicated by ellipses) omitted to heighten focus on feature ordering.

Because the first segment of code will be needed by every template that allows user-defined ordering, you'll probably want to put it in an include, rather than in the templates themselves.

include - vars

```
<!mtl "=== We want to pass forward most, but not all the values in the
URL.
          So we'll run through them using the weburl.arguments query
          & store the ones we want in a variable ===">
<mtl weburl.arguments>
     <mtl if {weburl.arguments.key<>'tid' and
weburl.arguments.key<>'temp_type' and weburl.arguments.key<>'tl' and
weburl.arguments.key<>'page' and weburl.arguments.value<>'' and
string.contains(weburl.arguments.key,"submit")=0 and
weburl.arguments.key<>"orderby"}>
          <mtl
$urlString="{$urlString}&{weburl.arguments.key}={weburl.arguments.value
fmt=url}">
     </mtl if>
</mtl weburl.arguments>
<!mtl "=== snip off leading ampersand ===">
<mtl $urlString=string.substr($urlString,2)>
```

/include - vars

rdetail

```
<mtl include name="include - vars">

<!mtl "using 2 vars here to preserve value passed in - for conditional
linking of column heads">
<mtl if {$orderby=''}>
     <mtl $theorderby="search_score,ad_type">
<mtl else>
     <mtl $theorderby="search_score,{$orderby},ad_type">
</mtl if>

<mtl classifieds.ad property=$property classification=$classification
category_number=$category_number ad_id=$ad_id show_text=y
keyword_query=$keyword keyword_font="<b>,</b>"  max_price=$adpricemax
min_price=$adpricemin no_data=continue blanks=" "
ads_per_page=$maxrec page=$page date=$date adnumber=$ad_number
ORDERBY=$theorderby>

<mtl if {classifieds.ad.index=1}>
     <table>
```

```
     <tr>
     <td>
     <mtl if val=$orderby contains="price">
          <mtl if {$orderby="price"}>
               <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=price:d" title="Sort, descending, by
Price">Price˘</a>
          <mtl else>
               <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=price" title="Sort by
Price">Price</a>
          </mtl if>
     <mtl else>
          <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=price" title="Sort by
Price">Price</a>
     </mtl if>

     </td><td align=center>

     <mtl if val=$orderby contains="feature_quantity(beds)">
          <mtl if {$orderby="feature_quantity(beds)"}>
               <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=feature_quantity(beds):d"
title="Sort, descending, by beds">Beds˘</a>
          <mtl else>
               <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=feature_quantity(beds)" title="Sort
by beds">Beds</a>
          </mtl if>
     <mtl else>
          <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=feature_quantity(beds):d"
title="Sort by beds">Beds</a>
     </mtl if>

     </td><td align=center>

     <mtl if val=$orderby contains="feature_quantity(baths)">
          <mtl if {$orderby="feature_quantity(baths)"}>
               <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=feature_quantity(baths):d"
title="Sort, descending, by Baths">Baths˘</a>
          <mtl else>
               <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=feature_quantity(baths)" title="Sort
by Baths">Baths</a>
          </mtl if>
     <mtl else>
          <a href="<mtl $baseref>?<mtl
$urlString>&temp_type=detail&orderby=feature_quantity(baths):d"
title="Sort by Baths">Baths</a>
     </mtl if>

     </td>
     <td valign="top">Description</td>
     </tr>
</mtl if>
```

...