CHAPTER 7: Using jQuery UI Autocomplete in Django Templates (P141 – 163)

In this chapter we will cover:

- jQuery UI's autocomplete and themeroller
- A "progressive enhancement" combobox
- What needs to be done on the server-side and client-side to do the following:
  - Using Django templating to dynamically create elements
  - Client-side event handling to send autocomplete-based selections to the server
  - DOM Level 0 and iframe-based alternatives on the client-side
  - Extending server-side Django Ajax views to handle updates from the client
  - Refining the working solution
- An example of practical problem solving when issues arise

jQuery UI's autocomplete and themeroller

- For further development, we will be using a **jQuery theme**.
- What specific theme to be used is customizable, but autocomplete and other plugins require some theme such as jQuery UI Themeroller provides.
- jQuery UI Themeroller, which lets you customize and tweak a theme (or just download a default), is available at: ***http://jqueryui.com/themeroller/***
- When you have made any customizations and downloaded a theme, you can unpack it under your static content directory. In our base.html template, after our site-specific stylesheet, we have added an include to the jQuery custom stylesheet.

```
{% block head_css_site %}<link rel="stylesheet" type="text/css"
href="/static/css/style.css" />
<link rel="stylesheet" type="text/css"
href="/static/css/smoothness/jquery-ui-1.8.2.custom.css" />
{% endblock head_css_site %}
```

- We will be using the jQuery UI combobox, which offers a "progressive enhancement" strategy by building a page that will still work with JavaScript off and will be more accessible than building a solution with nothing but Ajax.

Progressive enhancement

- "***Progressive enhancement***," in a nutshell, means that as much as possible you build a system that works without JavaScript or CSS, with semantic markup and similar practices, then add appearance with CSS, and then customize behavior with JavaScript.
- Here we follow the same basic formula for ***department***, ***location***, and ***reports_to***. We produce a list of all available options. The first entry is for no selected ***department/location/reports_to;*** as a courtesy to the user we add ***selected="selected"*** to the presently selected value so that the form is smart enough to remember the last selected value, rather than defaulting to a choice each time the user visits it.
- This much of the code is run once and creates the beginning of the containing paragraph, sets a ***strong*** tag, and creates the entry for no ***department*** selected:

```
<p>Department:
    <strong>
        <select name="department" id="department"
                class="autocomplete">
            <option
            {% if not entity.department.id %}
                selected="selected"
            {% endif %}
            value="department.-1">&mdash; Select &mdash;</option>

            {% for department in departments %}
            <option
            {% if department.id == entity.department.id %}
                selected="selected"
            {% endif %}
            value="department.{{ department.id }}">
                                  {{ department.name}}
            </option>
       {% endfor %}
        </select>
    </strong>
</p>
```

- Then we loop through the list of departments, creating an option that has a value of "department." followed by the id of the entity provided as a department. Remember that earlier we simply used a list of all entities for departments.
- If you are interested in tinkering, you could add a checkbox to indicate whether an entity should be considered a department or a ***reports_to*** candidate.

The `location` and `reports_to` are handled similarly:

```
Location:
    <select name="location" id="location" class="autocomplete">
```

We add the default, unselected option:

```
<option
{% if not entity.location.id %}
    selected="selected"
{% endif %}
value="location.-1">&mdash; Select &mdash;</option>
```

Then we loop through available locations and build their options.

```
    {% for location in locations %}
        <option
        {% if locationi.id == entity.locationi.id %}
            selected="selected"
        {% endif %}
        value="location.{{ location.id }}">
                        {{ location.identifier }}</option>
    {% endfor %}
    </select>
</p>
```

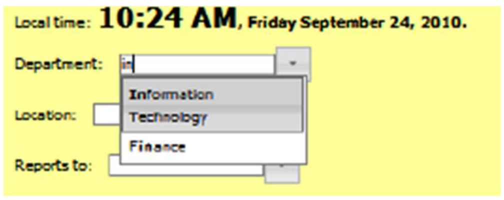
- And likewise in ***Reports to***:

```
<p>
Reports to:
    <select name="reports_to" id="reports_to" class="autocomplete">
        <option
        {% if not entity.reports_to.id %}
            selected="selected"
        {% endif %}
        value="reports_to_-1">&mdash; Select &mdash;</option>
        {% for reports_to in reports_to_candidates %}
            <option
            {% if reports_to.id == entity.reports_to.id %}
                selected="selected"
            {% endif %}
            value="reports_to_{{ reports_to.id }}">
                                 {{ reports_to.name }}</option>
        {% endfor %}
    </select>
</p>
```

- Our profile page is modified to provide more variables to the template. Later on, we might identify which entities can be an entity's departments and which entities can be reported to, thus improving the available options by paring away irrelevant entities, but for now we simply provide all entities.

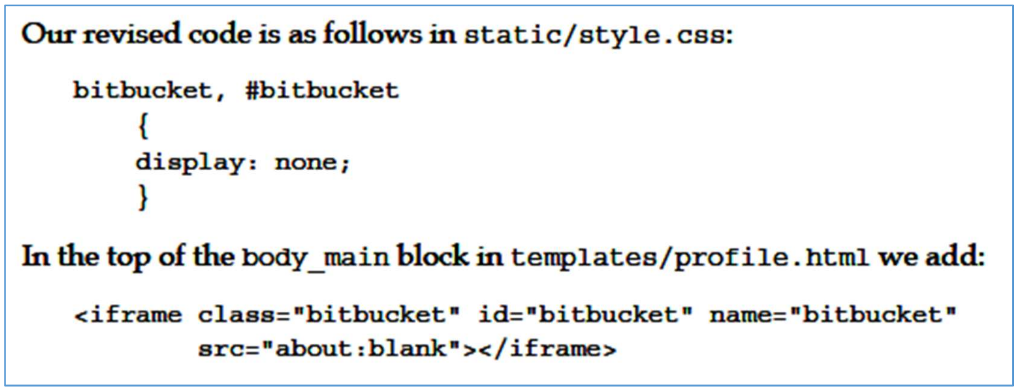
```
@login_required
def profile(request, id):
    entity = directory.models.Entity.objects.get(pk = id)
    emails = directory.models.EntityEmail.objects.filter(
             entity__exact = id).all()
    all_entities = directory.models.Entity.objects.all()
    all_locations = directory.models.Location.objects.all()
    return HttpResponse(get_template(u'profile.html').render(Context(
      {
      u'entity': entity,
      u'emails': emails,
      u'departments': all_entities,
      u'reports_to_candidates': all_entities,
      u'locations': all_locations,
      })))
```

- This provides nice-looking autocomplete functionality like:



A first workaround

- The obvious way for us to save the data is by an ***XMLHttpRequest*** based call but it is not strictly an interest to say that we go through ***XMLHttpRequest*** or ***jQuery*** for the submission. It would also work to have a form that submitted the same data to an ***iframe***.



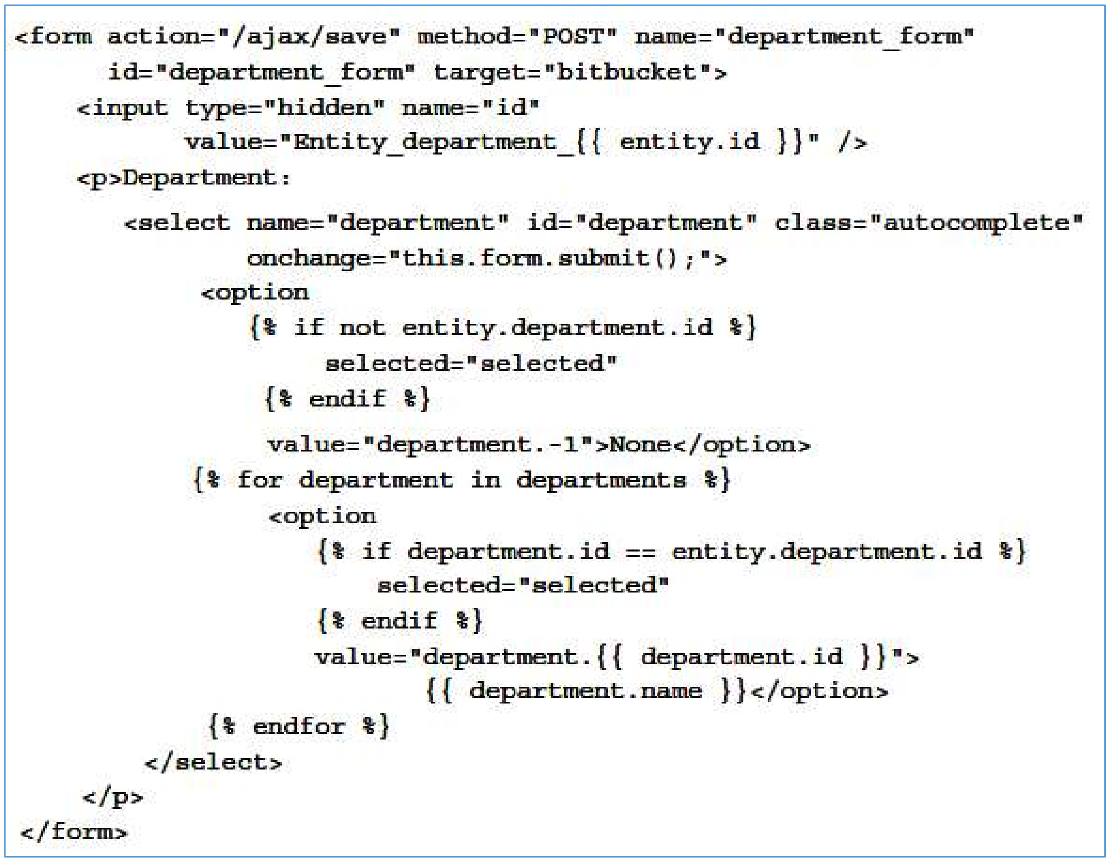
Our revised code is as follows in static/style.css:

```
bitbucket, #bitbucket
    {
    display: none;
    }
```

In the top of the body_main block in templates/profile.html we add:

```
<iframe class="bitbucket" id="bitbucket" name="bitbucket"
        src="about:blank"></iframe>
```

- The previous code is largely the same, but is wrapped in a form tag, and preceded by a hidden input designed to ensure that the view has all the information it needs.



```
<form action="/ajax/save" method="POST" name="department_form"
      id="department_form" target="bitbucket">
    <input type="hidden" name="id"
           value="Entity_department_{{ entity.id }}" />
    <p>Department:
        <select name="department" id="department" class="autocomplete"
                onchange="this.form.submit();">
            <option
               {% if not entity.department.id %}
                   selected="selected"
               {% endif %}
               value="department.-1">None</option>
           {% for department in departments %}
               <option
                  {% if department.id == entity.department.id %}
                      selected="selected"
                  {% endif %}
                  value="department.{{ department.id }}">
                      {{ department.name }}</option>
           {% endfor %}
        </select>
    </p>
</form>
```

- We define a link for the homepage:

```
<p>
    Homepage:
    {% if entity.homepage %}
        <a href="{{ entity.homepage }}">
    {% endif %}
    <strong class="edit_rightclick"
       id="entity_homepage_{{ entity.id }}">{{ entity.homepage }}
    </strong>
    {% if entity.homepage %}
        </a>
    {% endif %}
</p>
```

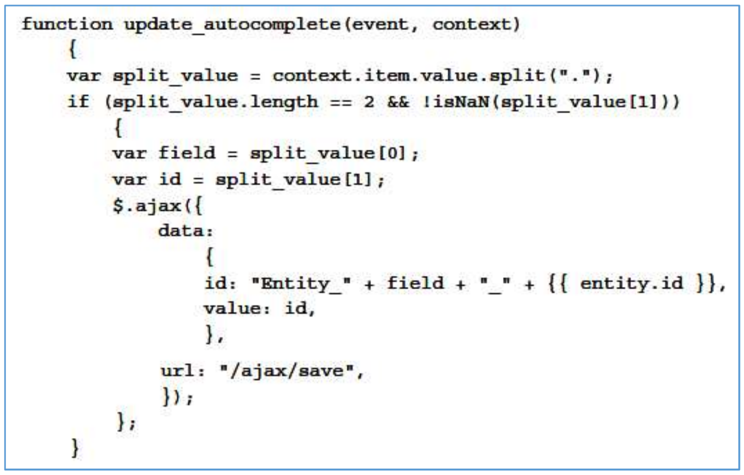
- We define similar, but not identical, handling for the e-mail:

```
<p>
    Email:
    <strong>
        {% for email in emails %}
            <a id="EntityEmail_email_{{ email.id }}"
               class="edit_rightclick"
               href="mailto:{{ email.email }}">
                {{ email.email }}</a>{% if not forloop.last %},
            {% endif %}
        {% endfor %}
        <span class="edit" id="EntityEmail_new_{{ entity.id }}">
            Click to add email.
        </span>
    </strong>
</p>
```

- The location fields, as the ***reports_to*** field in the following snippet, follow the same pattern as the department previously seen.
- We define a form:

```
<form action="/ajax/save" method="POST" name="location_form"
      id="location_form" target="bitbucket">
    <input type="hidden" name="id"
           value="Entity_location_{{ entity.id }}" />
    <p>
        Location:
        <select name="location" id="location" class="autocomplete"
          onchange="this.form.submit();">
         <option
         {% if not entity.location.id %}
            selected="selected"
         {% endif %}
         value="location.-1">None</option>
        {% for location in locations %}
           <option
            {% if locationi.id == entity.locationi.id %}
               selected="selected"
            {% endif %}
            value="location.{{ location.id }}">
                  {{ location.identifier }}</option>
        {% endfor %}
        </select>
    </p>
</form>
```

- The handler follows the signature for event handlers registered, as we wish to register them.
- However, we are developing with an analogous interface in mind: submitted data in the id-value format, with the ID following the ***ModelName_fieldname_instanceID*** naming convention.

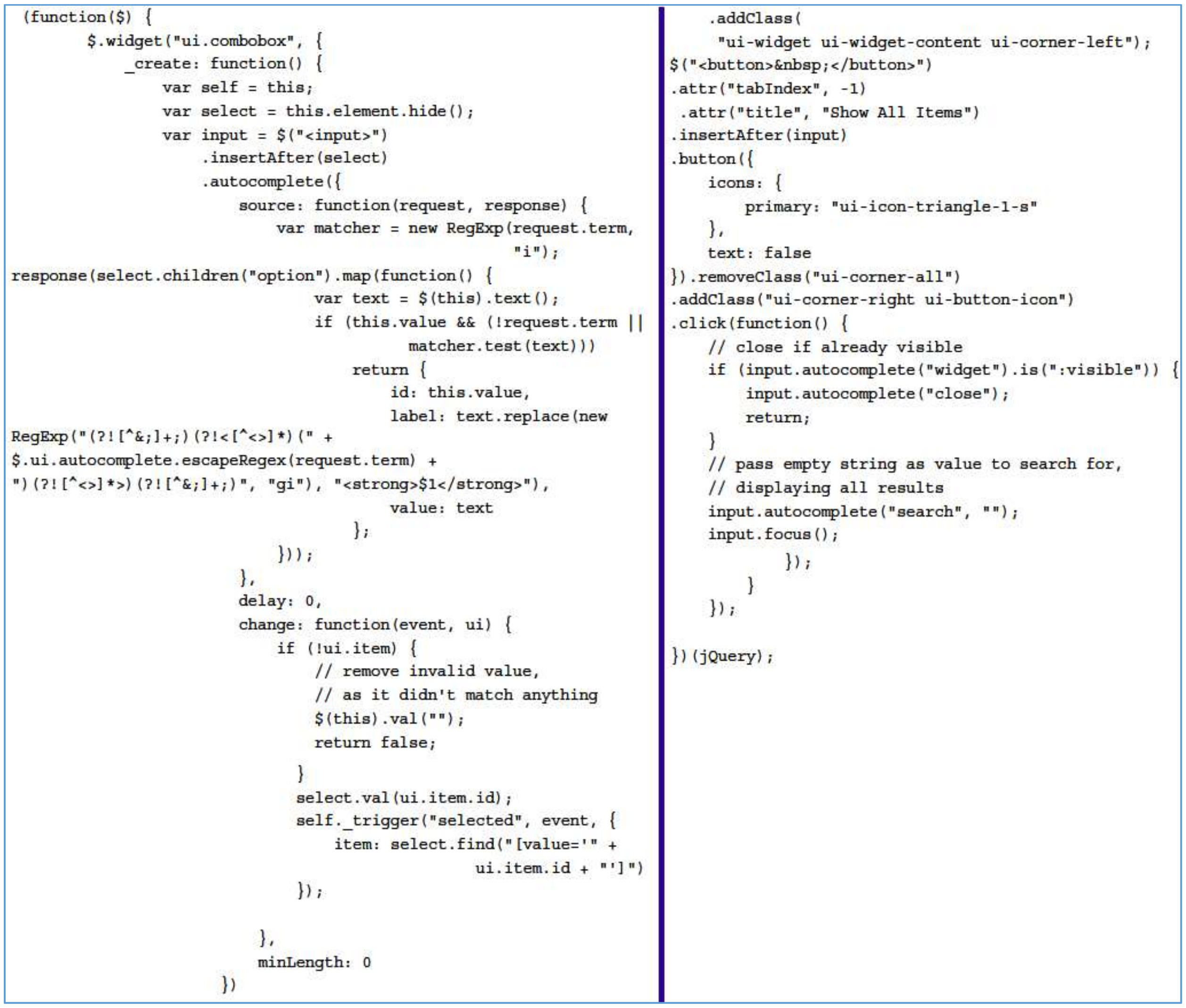


```
function update_autocomplete(event, context)
    {
    var split_value = context.item.value.split(".");
    if (split_value.length == 2 && !isNaN(split_value[1]))
        {
        var field = split_value[0];
        var id = split_value[1];
        $.ajax({
            data:
                {
                id: "Entity_" + field + "_" + {{ entity.id }},
                value: id,
                },

            url: "/ajax/save",
            });
        };
    }
```

Boilerplate code from jQuery UI documentation

- It is commonplace when using software to include adding boilerplate code. Here is an example. We insert boilerplate code from the documentation pages for jQuery UI at ***http://jqueryui.com/demos/autocomplete/#combobox:***



```
(function($) {
      $.widget("ui.combobox", {
           _create: function() {
                var self = this;
                var select = this.element.hide();
                var input = $("<input>")
                    .insertAfter(select)
                    .autocomplete({
                        source: function(request, response) {
                            var matcher = new RegExp(request.term,
                                                          "i");
response(select.children("option").map(function() {
                                var text = $(this).text();
                                if (this.value && (!request.term ||
                                          matcher.test(text)))
                                    return {
                                        id: this.value,
                                        label: text.replace(new
RegExp("(?![^&;]+;)(?!<[^<>]*)(" +
$.ui.autocomplete.escapeRegex(request.term) +
")(?![^<>]*>)(?![^&;]+;)", "gi"), "<strong>$1</strong>"),
                                        value: text
                                    };
                            }));
                        },
                        delay: 0,
                        change: function(event, ui) {
                            if (!ui.item) {
                                // remove invalid value,
                                // as it didn't match anything
                                $(this).val("");
                                return false;

                             }
                             select.val(ui.item.id);
                             self._trigger("selected", event, {
                                 item: select.find("[value='" +
                                                ui.item.id + "']")
                             });

                          },
                          minLength: 0
                      })
      .addClass(
       "ui-widget ui-widget-content ui-corner-left");
$("<button>&nbsp;</button>")
.attr("tabIndex", -1)
 .attr("title", "Show All Items")
.insertAfter(input)
.button({
    icons: {
        primary: "ui-icon-triangle-1-s"
    },
    text: false
}).removeClass("ui-corner-all")
.addClass("ui-corner-right ui-button-icon")
.click(function() {
    // close if already visible
    if (input.autocomplete("widget").is(":visible")) {
        input.autocomplete("close");
        return;
    }
    // pass empty string as value to search for,
    // displaying all results
    input.autocomplete("search", "");
    input.focus();
            });
        }
    });

})(jQuery);
```

Turning on Ajax behavior

- We make autocompletes out of the relevant selects, and also display the selects, which the ***combobox()*** call hides by default. The user now has a choice between the select and an autocomplete box.

```
$(function()
    {
    $(".autocomplete").combobox();
    $(".autocomplete").toggle();
```

- Here we have code which, from the documentation, might be expected to call the ***update_autocomplete()*** event handler when a selection or change is made.
- After this code, let's look at our updated code on the server side, and then see how this bend in the road can be addressed.

```
    /*
    $(".autocomplete").autocomplete({select: update_autocomplete});
    $(".autocomplete").bind({"autocompleteselect": update_
autocomplete});
    $(".autocomplete").bind({"autocompletechange": update_
autocomplete});
    */
    });
```

Code on the server side

- Here we have the updated ***save()*** view which has been expanded to address its broader scope. We accept either ***GET*** or ***POST*** requests, although requests that alter data should only be made by ***POST*** for production purposes, and save the dictionary for exploration.

```
@ajax_login_required
def save(request):
    try:
        html_id = request.POST[u'id']
        dictionary = request.POST
    except:
        html_id = request.GET[u'id']
        dictionary = request.GET
```

- If we have one of the autocomplete values, we have a hidden field named ***id*** which guarantees that any submission will have that field in its dictionary, either ***request.POST*** or ***request.GET***.
- However, the department, location, and ***reports_to*** fields are not all named value, and we manually check for them and use ***value*** as a default:

```
if html_id.startswith(u'Entity_department_'):
    value = dictionary[u'department']
elif html_id.startswith(u'Entity_location_'):
    value = dictionary[u'location']
elif html_id.startswith(u'Entity_reports_to_'):
    value = dictionary[u'reports_to']
else:
    value = dictionary[u'value']
```

- We perform some basic validation. Our code's HTML ID should only consist of word characters:

```
if not re.match(ur'^\w+$', html_id):
    raise Exception("Invalid HTML id.")
```

- Then we handle several special cases before the general-purpose code that handles most ***/ajax/save*** requests. If there is a new ***EntityEmail***, we create it and save it:

```
match = re.match(ur'EntityEmail_new_(\d+)', html_id)
if match:
    model = int(match.group(1))
    email = directory.models.EntityEmail(email = value, entity =
      directory.models.Entity.objects.get(pk = model))
    email.save()
    directory.functions.log_message(u'EntityEmail for Entity ' +
      str(model) + u' added by: ' + request.user.username + u',
      value: ' + value + u'\n')
    return HttpResponse(
      u'<a class="edit_rightclick"
           id="EntityEmail_email_' + str(email.id)
      + u'" href="mailto:' + value + u'">' + value + u'</a>' +
      u'''<span class="edit" id="EntityEmail_new_%s">
          Click to add email.</span>''' % str(email.id))
```

- Here we have the special case of an Entity's department being set.

```
elif html_id.startswith(u'Entity_department_'):
    entity_id = int(html_id[len(u'Entity_department_'):])
    department_id = int(value[len(u'department.'):])
    entity = directory.models.Entity.objects.get(pk = entity_id)
    if department_id == -1:
        entity.department = None
    else:
        entity.department = directory.models.Entity.objects.get(
                                pk = department_id)
    entity.save()
    return HttpResponse(value)
```

- The ***location*** code works the same way as ***reports_to*** and ***department***:

```
elif html_id.startswith(u'Entity_location_'):
    entity_id = int(html_id[len(u'Entity_location_'):])
    location_id = int(value[len(u'location.'):])
    if location_id == -1:
        entity.location = None
    else:
        entity.location = directory.models.Location.objects.get(
                              pk == location_id)
    entity.save()
    return HttpResponse(value)
```

```
elif html_id.startswith(u'Entity_reports_to_'):
    entity_id = int(html_id[len(u'Entity_reports_to'):])
    reports_to_id = int(value[len(u'reports_to.'):])
    entity = directory.models.Entity.object.get(pk = entity_id)
    if reports_to_id == -1:
        entity.reports_to = None
    else:
        entity.reports_to = directory.models.Entity.objects.get(
                                pk == reports_to_id)
    entity.save()
    return HttpResponse(value)
```

Refining our solution further

- We can set things up in the base template so that, for development, Django's informative error pages are displayed; we also set form submissions to ***POST*** by default. We make a target ***div*** for the notifications:

```
{% block body_site_announcements %}
{% endblock body_site_announcements %}
{% block body_notifications %}<div
    id="notifications"></div>
{% endblock body_notifications %}
```

- Then we add, to the ***footer_javascript_site*** block, a slightly tweaked ***send_notifications().***

```
<script language="JavaScript" type="text/javascript">
function send_notification(message)
    {
    $("#notifications").html("<p>" + message + "</p>");
    setTimeout("$('#notifications').show('slow').delay(" + (5000 +
      message.length * 2) + ").hide('slow');", 0);
    }
```

- Our notifications area has several different messages, not all of which need to be visually labelled as errors, so we move from a red-based styling to one that is silver and grey in ***static/css/style.css***:

```
#notifications
    {
    background-color: #c0c0c0;
    border: 3px solid #808080;
    display: none;
    padding: 20px;
    }
```

- We call, on page load, ***$.ajaxSetup()*** to specify a default error handler, and also specify form submission via ***POST***.

```
$(function()
    {
    $.ajaxSetup(
        {
        error: function(XMLHttpRequest, textStatus, errorThrown)
            {
            send_notification(XMLHttpRequest.responseText);
            },
        type: "POST",
        });
    });
</script>
```

- In the profile template, we will remove the containing form elements and the hidden inputs, and replace the contents of the ***onchange*** attributes.
- The Department ***paragraph*** now looks like a cross between the previous two entries.

```
<p>Department:
     <select name="department" id="department" class="autocomplete"
      onchange="update_autocomplete(
             'Entity_department_{{ entity.id}}', this.value);">
        <option
        {% if not entity.department.id %}
              selected="selected"
        {% endif %}
         value="department.-1">None</option>
        {% for department in departments %}
            <option
                {% if department.id == entity.department.id %}
                    selected="selected"
                {% endif %}
                value="department.{{ department.id }}">
                       {{ department.name }}</option>
                {% endfor %}
    </select>
</p>
```

- The ***location*** and ***reports_to*** areas follow suit:

```
<p>Location:
     <select name="location" id="location" class="autocomplete"
       onchange="update_autocomplete('Entity_location_{{ entity.id}}',
                                     this.value);">
     <option
     {% if not entity.location.id %}
         selected="selected"
     {% endif %}
     value="location.-1">None</option>
    {% for location in locations %}
        <option
        {% if location.id == entity.location.id %}
             selected="selected"
        {% endif %}
             value="location.{{ location.id }}">
                    {{ location.identifier }}</option>
    {% endfor %}
     </select>
</p>
```

- The ***Reports to*** field also follows suit:

```
<p>Reports to:
    <select name="reports_to" id="reports_to" class="autocomplete"
      onchange="update_autocomplete(
               'Entity_reports_to_{{ entity.id}}', this.value);">
   <option
   {% if not entity.reports_to.id %}
        selected="selected"
    {% endif %}
    value="reports_to_-1">None</option>
    {% for reports_to in reports_to_candidates %}
        <option
        {% if reports_to.id == entity.reports_to.id %}
            selected="selected"
        {% endif %}
        value="reports_to_{{ reports_to.id }}">
               {{ reports_to.name }}</option>
    {% endfor %}
     </select>
</p>
```

- For the autocomplete:

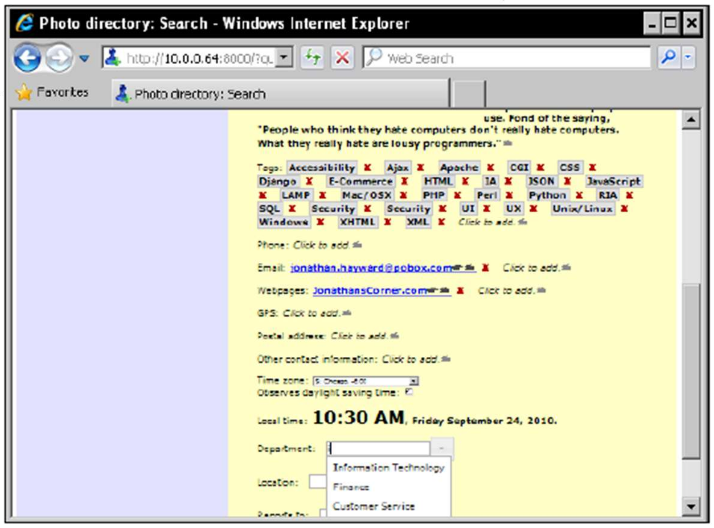


Photo directory: Search - Windows Internet Explorer

http://10.0.0.64:8000/?qu

Photo directory: Search

use. Fond of the saying,
"People who think they hate computers don't really hate computers.
What they really hate are lousy programmers."

Tags: Accessibility Ajax Apache CGI CSS Django E-Commerce HTML IA JSON JavaScript LAMP Mac/OSX PHP Perl Python RIA SQL Security Security UI UX Unix/Linux Windows XHTML XML Click to add.

Phone: Click to add.

Email: jonathan.hayward@pobox.com Click to add.

Webpages: JonathansCorner.com Click to add.

GPS: Click to add.

Postal address: Click to add.

Other contact information: Click to add.

Time zone: S. Chicago -6:00
Observes daylight saving time:

Local time: 10:30 AM, Friday September 24, 2010.

Department:
Information Technology
Finance
Customer Service

Location:

Reports to:

- Or, for another of several examples of user input that was allowed, here is an in-place edit used to add a new email address.

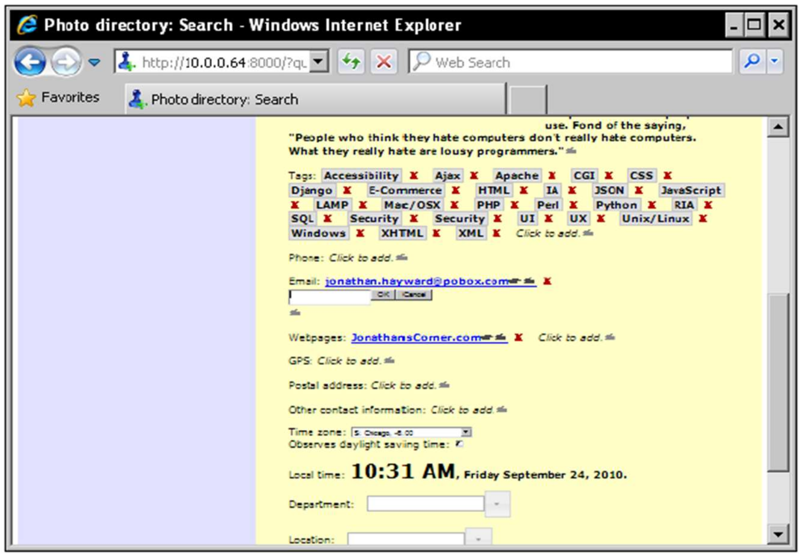


Photo directory: Search - Windows Internet Explorer

http://10.0.0.64:8000/?qu

Photo directory: Search

use. Fond of the saying,
"People who think they hate computers don't really hate computers.
What they really hate are lousy programmers."

Tags: Accessibility Ajax Apache CGI CSS Django E-Commerce HTML IA JSON JavaScript LAMP Mac/OSX PHP Perl Python RIA SQL Security Security UI UX Unix/Linux Windows XHTML XML Click to add.

Phone: Click to add.

Email: jonathan.hayward@pobox.com
OK Cancel

Webpages: JonathansCorner.com Click to add.

GPS: Click to add.

Postal address: Click to add.

Other contact information: Click to add.

Time zone: S. Chicago -6:00
Observes daylight saving time:

Local time: 10:31 AM, Friday September 24, 2010.

Department:

Location:
