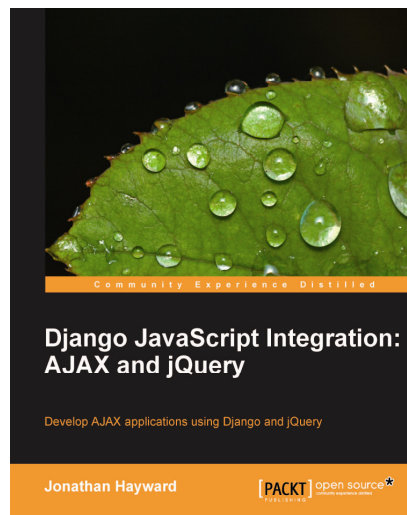


Django JavaScript Integration: AJAX and jQuery

Jonathan Hayward



Chapter No. 10

**"Tinkering Around: Bugfixes, Friendlier
Password Input, and a Directory That
Tells Local Time"**

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.10 "Tinkering Around: Bugfixes, Friendlier Password Input, and a Directory That Tells Local Time"

A synopsis of the book's content

Information on where to buy this book

About the Author

Jonathan Hayward as a child ranked 7th in a nationwide math contest, and later programmed a video game on his calculator. He holds master's degrees in bridging mathematics and computer science (UIUC), and philosophy and theology (Cambridge). Jonathan has lived in the U.S., Malaysia, France, and England, and has studied well over a dozen dialects and languages. He wears the hats of author, philosopher, theologian, artist, poet, wayfarer, philologist, inventor, and a skilled web developer who holds a deep interest in the human side of computing. He has a website showcasing his works at <http://JonathansCorner.com> and can be reached via e-mail at jonathan.hayward@pobox.com.

For More Information:

www.packtpub.com/django-javascript-integration-ajax-and-jquery/book

I would like to thank my parents, John and Linda, who love learning and taught me faith, my brothers, Matthew, Kirk and Joe, my parish, St. Innocent of Moscow, for a wealth of support. I would also like to thank the editorial team at Packt: Steven Wilding, who helped me come up with the book idea in the first place, Ved Prakash Jha, who helped see it to completion, and Joel Goyeva, who helped me with innumerable logistics along the way. And, of course, the reviewers Jake Kronika and Michael Szul, who offered an invaluable sharpening. The Django list, django-users@googlegroups.com, is worth its weight in gold. I would like to thank Daniel Roseman, Alex Robbins, Dan Harris, Karen Tracey, Oleg Lokalma, Mark Linsey, Jeff Green, Elijah Rutschman, Brian Neal, Euan Goddard, Sævar Öfjörð, "Ringemup", Ben Atkin, Tom Evans, Sam Lai, and Preston Holmes.

Authors have to leave somebody out who deserves to be mentioned; that's just part of the territory. But I would like to thank one person in particular: the reader. You're really the reason the book is here, and you've chosen to invest some money in a book and some time in fascinating technologies and let me help you along the way. Thank you so much.

For More Information:

www.packtpub.com/django-javascript-integration-ajax-and-jquery/book

Django JavaScript Integration: AJAX and jQuery

You want to create an AJAX application. Why would you use Django? Why would you use jQuery? Why would you use both together? Enter Django JavaScript Integration: AJAX and jQuery—your comprehensive answer to all these questions and the only extensive, practical, and hands-on guide to developing any AJAX application with Django and jQuery.

Gone are the days when you used to lament over the lack of official documentation on AJAX with Django. This book will teach you exactly why Django is called The web framework for perfectionists with deadlines, how jQuery—the "write less do more" JavaScript library—is practically a virtual higher-level language, and why they both deserve to be integrated with AJAX.

This hands-on-guide shows you how to put Django and jQuery together in the process of creating an AJAX application. In this book, they are brought together in a real-world scenario, with attention to usability, to build and develop an AJAX application.

The first two chapters provide a short and necessary introduction to the world of Django, jQuery, and AJAX; the remaining chapters are based on a case study that will make you realize the immense potential and benefits of integrating Django and jQuery with your AJAX application.

By the time you are done with this book, you'll be developing your AJAX applications with Django and jQuery in less time than you can say "integrate".

You will cover the basics of AJAX; use jQuery, the most common JavaScript library, on the client side, and learn form validation with an eye towards usability, build things with Django on the server side, handle login and authentication via Django-based AJAX, and then dip into the rich jQuery plugin ecosystem to build in-place editing into your pages.

You will add auto-complete functionality courtesy of jQuery UI, easily build forms with Django ModelForm, and then look at a client-side search implementation that can look things up without network access after initial download. You will learn to implement a simple, expandable undo system, and offer more full-blooded account management, tinker, fix some bugs, offer a more usable way to handle password input, add local time support for people who are not in your time zone, look at usability, and finally take a look at debugging.

After working through this book, you will have both an AJAX application: a Web 2.0 employee intranet photo directory, and with it a deep understanding that you can use to customize, extend, and further develop it in your organization.

For More Information:

www.packtpub.com/django-javascript-integration-ajax-and-jquery/book

What This Book Covers

This book covers Django JavaScript integration and building an Ajax application with Django on the server side and jQuery on the client side. It provides first an overview, then a first Ajax application, and introduces jQuery; discusses form validation, server-side database search; Ajax login facilities; jQuery in-place editing and autocomplete, Django ModelForm, and how to give auto-generated forms a transformational CSS makeover. It also discusses client-side functionality, customization, and further development with tinkering and added features, before a grand finale exploring usability, and an appendix on debugging hard JavaScript bugs.

Chapter 1, jQuery and Ajax Integration in Django lays a solid foundation and introduces you to the working pieces of Django Ajax to be explored in the rest of the book.

Chapter 2, jQuery—the Most Common JavaScript Framework explores the "higher-level" way of doing things in jQuery. You will learn how jQuery is not Python and does not look like Python, but how there is something "Pythonic in spirit" about how it works.

Chapter 3, Validating Form Input on the Server Side will teach you how to send an Ajax request to the server via jQuery, and validate it on the server side based on the principle that all input is guilty until proven innocent of being malicious, malformed, incomplete, or otherwise invalid.

Chapter 4, Server-side Database Search with Ajax looks both at the merits of handling searching and other backend functions with the full power of a backend environment, and explores why, on the client side, you should work hard to be as lazy as possible in doing network-related work.

Chapter 5, Signing-up and Logging into a Website Using Ajax introduces Django authentication facilities and account management and includes both server-side and client-side code.

Chapter 6, jQuery In-place Editing Using Ajax goes from a basic foundation to a continuing practical application. It will show a way to use jQuery to make an in-place replacement of a table that allows in-place editing, which communicates with the server in the background, adding persistence to changes.

For More Information:

www.packtpub.com/django-javascript-integration-ajax-and-jquery/book

Chapter 7, Using jQuery UI Autocomplete in Django Templates tells you what you need on the client side and server side to get autocomplete working with jQuery UI. It also includes creative problem solving when something goes wrong. This chapter will tell you why it is not uncommon for programmers to write plugins their first day doing jQuery.

Chapter 8, Django ModelForm: a CSS Makeover explores Django ModelForm and how to use it.

Chapter 9, Database and Search Handling covers all the bases for a simple, AJAX solution. In addition to showing "lazy" best practices, it also showcases a JavaScript in-memory database, with an application designed, at the developer's preference, to either always perform lazy handling of search and other requests, or start loading an in-memory database and falling back to lazy handling until the in-memory database is available.

Chapter 10, Tinkering Around: Bugfixes, Friendlier Password Input, and a Directory That Tells Local Time covers some tinkering and tweaks, and bugfixes along the way

Chapter 11, Usability for Hackers steps back from your application and takes a look at usability and the bedrock competencies hackers can leverage to do usability.

Appendix, Debugging Hard JavaScript Bugs looks at the state of mind that is needed to debug difficult bugs.

For More Information:

www.packtpub.com/django-javascript-integration-ajax-and-jquery/book

10

Tinkering Around: Bugfixes, Friendlier Password Input, and a Directory That Tells Local Time

One of the great joys of programming is not when we are trying to get the bare essentials basically working, but when the system is working as a whole, and we start to ask, "What about this? What about that?" One positive sense of the term "hacking" can refer to this tinkering, and it can be a joy to tinker with an already working system to see what enhancements are possible. Here we will tinker with the system and make some minor tweaks and two slightly more major enhancements. We will cover:

- Minor bugfixes and enhancements
- A more usable input solution for passwords
- Telling an (approximate) local time for other people we are working with, who may be in different time zones

Let's dig in.

Minor tweaks and bugfixes

Good tinkering can be a process that begins with tweaks and bugfixes, and snowballs from there. Let's begin with some of the smaller tweaks and bugfixes before tinkering further.

For More Information:

www.packtpub.com/django-javascript-integration-ajax-and-jquery/book

Setting a default name of "(Insert name here)"

Most of the fields on an Entity default to blank, which is in general appropriate. However, this means that there is a zero-width link for any search result which has not had a name set. If a user fills out the Entity's name before navigating away from that page, everything is fine, but it is a very suspicious assumption that all users will magically use our software in whatever fashion would be most convenient for our implementation.

So, instead, we set a default name of "(Insert name here)" in the definition of an Entity, in `models.py`:

```
name = models.TextField(blank = True,
    default = u'(Insert name here)')
```

Eliminating Borg behavior

One variant on the classic Singleton pattern in **Gang of Four** is the **Borg** pattern, where arbitrarily many instances of a Borg class may exist, but they share the same dictionary, so that if you set an attribute on one of them, you set the attribute on all of them. At present we have a bug, which is that our views pull all available instances. We need to specify something different. We update the end of `ajax_profile()`, including a slot for time zones to be used later in this chapter, to:

```
return render_to_response(u'profile_internal.html',
    {
        u'entities': directory.models.Entity.objects.filter(
            is_invisible = False).order_by(u'name'),
        u'entity': entity,
        u'first_stat': directory.models.Status.objects.filter(
            entity = id).order_by(
                u'-datetime')[directory.settings.INITIAL_STATI],
        u'gps': gps,
        u'gps_url': gps_url,
        u'id': int(id),
        u'emails': directory.models.Email.objects.filter(
            entity = entity, is_invisible = False),
        u'phones': directory.models.Phone.objects.filter(
            entity = entity, is_invisible = False),
        u'second_stat': directory.models.Status.objects.filter(
            entity = id).order_by(
                u'-datetime')[directory.settings.INITIAL_STATI:],
        u'tags': directory.models.Tag.objects.filter(entity = entity,
            is_invisible = False).order_by(u'text'),
```



```

u'time_zones': directory.models.TIME_ZONE_CHOICES,
u'urls': directory.models.URL.objects.filter(entity = entity,
is_invisible = False),
})

```

Likewise, we update `homepage()`:

```

profile = template.render(Context(
{
u'entities':
    directory.models.Entity.objects.filter(
        is_invisible = False),
u'entity': entity,
u'first_statu': directory.models.Status.objects.filter(
    entity = id).order_by(
        u'-datetime')[directory.settings.INITIAL_STATI],
u'gps': gps,
u'gps_url': gps_url,
u'id': int(id),
u'emails': directory.models.Email.objects.filter(
    entity = entity, is_invisible = False),
u'phones': directory.models.Phone.objects.filter(
    entity = entity, is_invisible = False),
u'query': urllib.quote(query),
u'second_statu': directory.models.Status.objects.filter(
    entity = id).order_by(
        u'-datetime')[directory.settings.INITIAL_STATI:],
u'time_zones': directory.models.TIME_ZONE_CHOICES,
u'tags': directory.models.Tag.objects.filter(
    entity = entity,
    is_invisible = False).order_by(u'text'),
u'urls': directory.models.URL.objects.filter(
    entity = entity, is_invisible = False),
}))

```

Confusing jQuery's `load()` with `html()`

If we have failed to load a profile in the main `search.html` template, we had a call to `load("")`. What we needed was:

```

else
{
$("#profile").html("");
}

```

`$("#profile").load("")` loads a copy of the current page into the `div` named `profile`. We can improve on this slightly to "blank" contents that include the default header:

```
else
{
$("#profile").html("<h2>People, etc.</h2>");
}
```

Preventing display of deleted instances

In our system, enabling undo means that there can be instances (Entities, Emails, URLs, and so on) which have been deleted but are still available for undo. We have implemented deletion by setting an `is_invisible` flag to `True`, and we also need to check before displaying to avoid puzzling behavior like a user deleting an Entity, being told **Your change has been saved**, and then seeing the Entity's profile displayed exactly as before.

We accomplish this by specifying, for a Queryset `.filter(is_invisible = False)` where we might earlier have specified `.all()`, or adding `is_invisible = False` to the conditions of a pre-existing filter; for instance:

```
def ajax_download_model(request, model):
    if directory.settings.SHOULD_DOWNLOAD_DIRECTORY:
        json_serializer = serializers.get_serializer(u'json')()
        response = HttpResponse(mimetype = u'application/json')
        if model == u'Entity':
            json_serializer.serialize(getattr(directory.models,
                model).objects.filter(
                    is_invisible = False).order_by(u'name'),
                ensure_ascii = False, stream = response)
        else:
            json_serializer.serialize(getattr(directory.models,
                model).objects.filter(is_invisible = False),
                ensure_ascii = False,
                stream = response)
        return response
    else:
        return HttpResponse(u'This feature has been turned off.')
```

In the main view for the profile, we add a check in the beginning so that a (basically) blank result page is shown:

```
def ajax_profile(request, id):
    entity = directory.models.Entity.objects.filter(id = int(id))[0]
    if entity.is_invisible:
        return HttpResponse(u'<h2>People, etc.</h2>')
```

One nicety we provide is usually loading a profile on mouseover for its area of the search result page. This means that users can more quickly and easily scan through drilldown pages in search of the right match; however, there is a performance gotcha for simply specifying an onmouseover handler. If you specify an onmouseover for a containing div, you may get a separate event call for every time the user hovers over an element contained in the div, easily getting 3+ calls if a user moves the mouse over to the link. That could be annoying to people on a VPN connection if it means that they are getting the network hits for numerous needless profile loads.

To cut back on this, we define an initially null variable for the last profile moused over:

```
PHOTO_DIRECTORY.last_mouseover_profile = null;
```

Then we call the following function in the containing div element's onmouseover:

```
PHOTO_DIRECTORY.mouseover_profile = function(profile)
{
    if (profile != PHOTO_DIRECTORY.last_mouseover_profile)
    {
        PHOTO_DIRECTORY.load_profile(profile);
        PHOTO_DIRECTORY.last_mouseover_profile = profile;
        PHOTO_DIRECTORY.register_editables();
    }
}
```

The relevant code from `search_internal.html` is as follows:

```
<div class="search_result"
    onmouseover="PHOTO_DIRECTORY.mouseover_profile(
        {{ result.id }});"
    onclick="PHOTO_DIRECTORY.click_profile({{ result.id }});">
```

We usually, but not always, enable this mouseover functionality; not always, because it works out to annoying behavior if a person is trying to edit, does a drag select, mouses over the profile area, and reloads a fresh, non-edited profile. Here we edit the Jeditable plugin's source code and add a few lines; we also perform a second check for if the user is logged in, and offer a login form if so:

```
/* if element is empty add something clickable
   (if requested) */
if (!$.trim($(this).html())) {
    $(this).html(settings.placeholder);
}
```

```
$(this).bind(settings.event, function(e) {

    $("div").removeAttr("onmouseover");
    if (!PHOTO_DIRECTORY.check_login())
    {
        PHOTO_DIRECTORY.offer_login();
    }
    /* abort if disabled for this element */
    if (true === $(this).data('disabled.editable')) {
        return;
    }
}
```

For Jeditable-enabled elements, we can override the placeholder for an empty element at method call, but the default placeholder is cleared when editing begins; overridden placeholders aren't. We override the placeholder with something that gives us a little more control and styling freedom:

```
// publicly accessible defaults
$.fn.editable.defaults = {
    name      : 'value',
    id        : 'id',
    type      : 'text',
    width     : 'auto',
    height    : 'auto',
    event     : 'click.editable',
    onblur    : 'cancel',
    loadtype  : 'GET',
    loadtext  : 'Loading...',
    placeholder: '<span class="placeholder">
                Click to add.</span>',
    loaddata  : {},
    submitdata : {},
    ajaxoptions: {}
};
```

All of this is added to the file `jquery.jeditable.js`.

We now have, as well as an `@ajax_login_required` decorator, an `@ajax_permission_required` decorator. We test for this variable in the default postprocessor specified in `$.ajaxSetup()` for the complete handler. Because Jeditable will place the returned data inline, we also refresh the profile.

This occurs after the code to check for an undoable edit and offer an undo option to the user.

```
complete: function(XMLHttpRequest, textStatus)
{
    var data = XMLHttpRequest.responseText;
```

```

var regular_expression = new RegExp("<!--" +
    "-# (\\d+) #-" + "->");
if (data.match(regular_expression))
{
    var match = regular_expression.exec(data);
    PHOTO_DIRECTORY.undo_notification(
        "Your changes have been saved. " +
        "<a href='JavaScript:PHOTO_DIRECTORY.undo(" +
        match[1] + ")'>Undo</a>");
}
else if (data == '{"not_permitted": true}' ||
    data == '{"not_permitted": true}')
{
    PHOTO_DIRECTORY.send_notification(
        "We are sorry, but we cannot allow you " +
        "to do that.");
    PHOTO_DIRECTORY.reload_profile();
}
},

```

Note that we have tried to produce the least painful of clear message we can: we avoid both saying "You shouldn't be doing that," and a terse, "bad movie computer"-style message of "Access denied" or "Permission denied."

We also removed from that method code to call `offer_login()` if a call came back not authenticated. This looked good on paper, but our code was making Ajax calls soon enough that the user would get an immediate, unprovoked, modal login dialog on loading the page.

Adding a favicon.ico

In terms of minor tweaks, some visually distinct `favicon.ico` (<http://softpedia.com/> is one of many free sources of `favicon.ico` files, or the favicon generator at <http://tools.dynamicdrive.com/favicon/> which can take an image like your company logo as the basis for an icon) helps your tabs look different at a glance from other tabs. Save a good, simple favicon in `static/favicon.ico`. The icon may not show up immediately when you refresh, but a good favicon makes it slightly easier for visitors to manage your pages among others that they have to deal with. It shows up in the address bar, bookmarks, and possibly other places.

This brings us to the end of the minor tweaks; let us look at two slightly larger additions to the directory.

Handling password input in a slightly different way

Our first addition has to do with password inputs. The traditional style of password input leaves plenty of room for second guessing about "Did I hit this key hard enough? Did I mistype something?" Logging in and specifying your password the traditional way ranks up with CAPTCHA as the hardest part of the form for regular users (apart from any disability issues).

What we will do, then, is present a regular text input for the default (users can click on **Hide password** for the old-school password input), and work a little Ajax to let the users switch. We will have two inputs, and when one of them receives a keydown or keyup event, its data is copied to the other. Only one of them, and only one link, will be visible at a time. In our `style.css` we have:

```
#new_password_hidden, #password_hidden
{
    display: none
}

#show_new_password, #show_password
{
    display: none;
}
```

In the template, we expand the markup for the login and create account forms:

```
<div id="login_form" title="Log in">
  <form>
    <fieldset>
      <label for="login">Login</label><br />
      <input type="text" name="login" id="login"
        class="text ui-widget-content ui-corner-all" /><br />
      <label for="password">Password</label><br />
      <input onkeyup="PHOTO_DIRECTORY.field_sync(
        'password_visible', 'password_hidden');"
        onkeydown="PHOTO_DIRECTORY.field_sync(
        'password_visible', 'password_hidden');"
        autocomplete="off" type="text" name="password_visible"
        id="password_visible"
        class="text ui-widget-content ui-corner-all" />
      <input onkeyup="PHOTO_DIRECTORY.field_sync(
        'password_hidden', 'password_visible');"
        onkeydown="PHOTO_DIRECTORY.field_sync(
        'password_hidden', 'password_visible');"
        type="password" name="password_hidden"
        id="password_hidden"
        class="text ui-widget-content ui-corner-all" />
    </fieldset>
  </form>
</div>
```

```

        type="password" name="password_hidden"
        id="password_hidden"
        class="text ui-widget-content ui-corner-all" /><br />
<a id="hide_password" name="hide_password"
    href="JavaScript:PHOTO_DIRECTORY.hide_element(
        'password');">Hide password</a>
<a id="show_password" name="show_password"
    href="JavaScript:PHOTO_DIRECTORY.show_element(
        'password');">Show password</a>
    </fieldset>
</form>
</div>
<div id="create_account" title="Create account">
    <form>
        <fieldset>
            <label for="new_username">Account name</label><br />
            <input type="text" name="new_username" id="new_username"
                class="text ui-widget-content ui-corner-all" /><br />
            <label for="new_email">Email</label><br />
            <input type="text" name="new_email" id="new_email"
                class="text ui-widget-content ui-corner-all" /><br />
            <label for="new_password">Password</label><br />
            <input onkeyup="PHOTO_DIRECTORY.field_sync(
                'new_password_visible', 'new_password_hidden');"
                onkeydown="PHOTO_DIRECTORY.field_sync(
                'new_password_visible', 'new_password_hidden');"
                autocomplete="off" type="text"
                name="new_password_visible" id="new_password_visible"
                class="text ui-widget-content ui-corner-all" />
            <input onkeyup="PHOTO_DIRECTORY.field_sync(
                'new_password_hidden', 'new_password_visible');"
                onkeydown="PHOTO_DIRECTORY.field_sync(
                'new_password_hidden', 'new_password_visible');"
                type="password" name="new_password_hidden"
                id="new_password_hidden"
                class="text ui-widget-content ui-corner-all" /><br />
            <a id="hide_new_password" name="hide_new_password"
                href="JavaScript:PHOTO_DIRECTORY.hide_element(
                    'new_password');">Hide password</a>
            <a id="show_new_password" name="show_new_password"
                href="JavaScript:PHOTO_DIRECTORY.show_element(
                    'new_password');">Show password</a>
        </fieldset>
    </form>
</div>

```

With this change, we also change the `search.html` file's reference to `document.getElementById("password").value` to `document.getElementById("password_visible").value`.

`field_sync()` copies data from one specified form element to another:

```
PHOTO_DIRECTORY.field_sync = function(from, to)
{
    $("#" + to).val($("#" + from).val());
}
```

`hide_element()` and `show_element()` feed their argument into toggling visibility for several elements:

```
PHOTO_DIRECTORY.hide_element = function(name)
{
    $("#hide_" + name + "." + name + "_visible").hide();
    $("#show_" + name + "." + name + "_hidden").show();
}
```

And with the last arguments reversed:

```
PHOTO_DIRECTORY.show_element = function(name)
{
    $("#hide_" + name + "." + name + "_visible").show();
    $("#show_" + name + "." + name + "_hidden").hide();
}
```

In our HTML markup for the inputs of type text, we specified `autocomplete="off"`, which is an important housekeeping detail to exclude the password from the list of form elements that are quietly recorded whether the user wants it that way or not.

And that completes this usability enhancement.

A directory that includes local timekeeping

We live in a world where we can deal with people in many different time zones, and sometimes it would be nice to know what time it is for the other person.

The solution here is an approximate solution that can run aground on the intricacies of Daylight Saving Time. Readers wishing for a more accurate solution may find an Ajax use of the Python module `pytz` on the backend (<http://pytz.sourceforge.net/>) to be more accurate in handling Daylight Saving Time. All the world's time zones amount to a lot of nooks and crannies, and a server-side database allows a finer granularity than a quick client-side solution. However, even an approximate client-side solution can be helpful in knowing, "What time of day is it for the other person?"

First of all, we define a choice for a time zone. (Note that we are using strings rather than decimals for offsets; pure decimals had implementation issues and produced results that JavaScript did not recognize as equal.)

```
TIME_ZONE_CHOICES = (
    (None, "Select"),
    ("1.0", "A: Paris, +1:00"),
    ("2.0", "B: Athens, +2:00"),
    ("3.0", "C: Moscow, +3:00"),
    ("4.0", "D: Dubai, +4:00"),
    ("4.5", "-: Kabul, +4:30"),
    ("5.0", "E: Karachi, +5:00"),
    ("5.5", "-: New Delhi, +5:30"),
    ("5.75", "-: Kathmandu, :5:45"),
    ("6.0", "F: Dhaka, +6:00"),
    ("6.5", "-: Rangoon, +6:30"),
    ("7.0", "G: Jakarta, +7:00"),
    ("8.0", "H: Kuala Lumpur, +8:00"),
    ("9.0", "I: Tokyo, +9:00"),
    ("9.5", "-: Adelaide, +9:30"),
    ("10.0", "K: Sydney, +10:00"),
    ("10.5", "-: Lord Howe Island, +10:30"),
    ("11.0", "L: Solomon Islands, +11:00"),
    ("11.5", "-: Norfolk Island, +11:50"),
    ("12.0", "M: Auckland, +12:00"),
    ("12.75", "-: Chatham Islands, +12:45"),
    ("13.0", "-: Tonga, +13:00"),
    ("14.0", "-: Line Islands, +14:00"),
    ("-1.0", "N: Azores, -1:00"),
    ("-2.0", "O: Fernando de Norohna, -2:00"),
    ("-3.0", "P: Rio de Janiero, -3:00"),
    ("-3.5", "-: St. John's, -3:50"),
    ("-4.0", "Q: Santiago, -4:00"),
    ("-4.5", "-: Caracas, -4:30"),
    ("-5.0", "R: New York City, -5:00"),
    ("-6.0", "S: Chicago, -6:00"),
    ("-7.0", "T: Boulder, -7:00"),
    ("-8.0", "U: Los Angeles, -8:00"),
    ("-9.0", "V: Anchorage, -9:00"),
    ("-9.5", "-: Marquesas Islands, -9:30"),
    ("-10.0", "W: Hawaii, -10:00"),
    ("-11.0", "X: Samoa, -11:00"),
    ("-12.0", "Y: Baker Island, -12:00"),
    ("0.0", "Z: London, +0:00"),
)
```

We expand the Entity definition to include a time zone slot, along with a checkmark for whether Daylight Saving Time is observed:

```
class Entity(models.Model):
    active = models.BooleanField(blank = True)
    department = models.ForeignKey(u'self', blank = True, null =
        True, related_name = u'member')
    description = models.TextField(blank = True)
    gps = GPSField()
    image_mimetype = models.TextField(blank = True, null = True)
    is_invisible = models.BooleanField(default = False)
    location = models.ForeignKey(u'self', blank = True, null = True,
        related_name = u'occupant')
    name = models.TextField(blank = True,
        default = u'(Insert name here)')
    observes_daylight_saving_time = models.BooleanField(
        blank = True, default = True)
    other_contact = models.TextField(blank = True)
    postal_address = models.TextField(blank = True)
    publish_externally = models.BooleanField(blank = True)
    reports_to = models.ForeignKey(u'self', blank = True,
        null = True, related_name = u'subordinate')
    start_date = models.DateField(blank = True, null = True)
    time_zone = models.CharField(max_length = 5, null = True,
        choices = TIME_ZONE_CHOICES)
    title = models.TextField(blank = True)
    class Meta:
        permissions = (
            ("view_changelog", "View the editing changelog"),
        )
```

Then we expand the `profile_internal.html` template to allow selection of time zone and Daylight Saving Time observance:

```
<p>Time zone:
<select name="time_zone" id="time_zone"
onchange="PHOTO_DIRECTORY.update_autocomplete(
'Entity_time_zone_{{ id }}', 'time_zone');">
    {% for time_zone in time_zones %}
        <option value="{{ time_zone.0 }}"
            {% if time_zone.0 == entity.time_zone %}
                selected="selected"
            {% endif %}
        >{{ time_zone.1 }}</option>
    {% endfor %}
```

```

        </select><br />
    Observes daylight saving time: <input type="checkbox"
        name="observes_daylight_saving_time"
        id="observes_daylight_saving_time"
        onchange="PHOTO_DIRECTORY.update_autocomplete(
            'Entity_observes_daylight_saving_time_{{ id }}',
            'observes_daylight_saving_time');"
        {% if entity.observes_daylight_saving_time %}
            checked="checked"
        {% endif %}
    /></p>

```

And in addition, we add hooks to be filled out with the local time:

```

    {% if entity.time_zone != None %}
    <p>Local time:
    <span id="local_time_zone">{{ entity.time_zone }}</span>
    <span id="local_time"></span></p>
    {% endif %}

```

The `local_time_zone` span is hidden:

```

#local_time_zone
{
    display: none;
}

```

Its purpose is to let Ajax fetch the Entity's time zone. In `search.html`, we expand the `register_update()` to set the clock:

```

PHOTO_DIRECTORY.register_update = function()
{
    PHOTO_DIRECTORY.limit_width("img.profile", 150);
    PHOTO_DIRECTORY.limit_width("img.search_results", 80);
    PHOTO_DIRECTORY.register_editables();
    PHOTO_DIRECTORY.register_autocomplete();
    if (!PHOTO_DIRECTORY.SHOULD_TURN_ON_HIJAXING)
    {
        $("a").removeAttr("onclick");
        // This link needs to be hijaxed:
        $("#add_new").click(function()
        {
            PHOTO_DIRECTORY.add_new();
            return false;
        });
    }
}

```

One gotcha has to do with Daylight Saving Time: some places observe daylight saving time, some places don't, and if you don't take it into account, it's an easy way to be wrong by an hour about the other party's local time.

JavaScript offers no explicit facility to tell if our local time includes Daylight Saving Time. One way to tell if we have Daylight Saving Time right now is by looking at our offset from UTC and seeing if it is the same as the January 1 offset. If the two offsets are different then we are in Daylight Saving Time.

```
if ($("#local_time_zone").html())
{
    var date = new Date();
    var profile_offset = -parseFloat(
        $("#local_time_zone").html()) * 3600000;
    var january_offset = new Date(date.getFullYear(), 0,
        1).getTimezoneOffset() * 60000;
    var our_offset = date.getTimezoneOffset() * 60000;
    if (our_offset != january_offset)
    {
        if (document.getElementById(
            "observes_daylight_saving_time").checked)
        {
            profile_offset -= 60 * 60 * 1000;
        }
    }
    PHOTO_DIRECTORY.update_clock(our_offset - profile_offset,
        PHOTO_DIRECTORY.current_profile);
}
```

And we define the `update_clock` function. First it checks to see if the profile it was registered with is the current profile, and if not, we bail out:

```
PHOTO_DIRECTORY.update_clock = function(offset, id)
{
    if (id != PHOTO_DIRECTORY.current_profile)
    {
        return;
    }
}
```

Then we make an adjusted date, "fudged" to give us values for the Entity's local time, and build human-friendly display output for it.

```
var adjusted_date = new Date(new Date().getTime() + offset);
var days = ["Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"];
```

```

var months = ["January", "February", "March", "April",
    "May", "June", "July", "August", "September", "November",
    "December"];
var ampm = "AM";
var hours = adjusted_date.getHours()
if (hours > 11)
{
    hours -= 12;
    ampm = "PM"
}
if (hours == 0)
{
    hours = 12;
}
var formatted_date = "<strong>" + hours + ":";
if (adjusted_date.getMinutes() < 10)
{
    formatted_date += "0";
}
formatted_date += adjusted_date.getMinutes() + " " + ampm;
formatted_date += "</strong> ";
formatted_date += days[adjusted_date.getDay()] + " ";
formatted_date += months[adjusted_date.getMonth()] + " ";
formatted_date += adjusted_date.getDate() + ", ";
formatted_date += adjusted_date.getFullYear();
$("#local_time").html(formatted_date + ".");

```

An example of the formatted output this produces is "12:01 PM, January 1, 2001."

Lastly, before closing, we set the clock to update again in a second, to keep the displayed value fresh.

```

setTimeout("PHOTO_DIRECTORY.update_clock(" + offset + ", " +
    id + ")", 1000);
}

```

In the code to dynamically build a profile, we add, after **Other contact information** and before **Department**, fields to build the equivalent of what we added to `profile_internal.html` earlier:

```

result += "<p>Other contact information: <strong ";
result += "class='edit_textarea' " +
    "title='Click to edit.' " +
    "id='Entity_other_contact_" + id + "'> " +
    entity.fields.other_contact + "</strong></p>\n";

```

```
result += "<p>Time zone: ";
result += "<select name='time_zone' id='time_zone'";
result += "onchange="
    'PHOTO_DIRECTORY.update_autocomplete(
        \"Entity_time_zone_{{ id }}\", \"time_zone\");'>";
{% for time_zone in time_zones %}
    result += "<option value='{{ time_zone.0 }}'>";
    result += "{{ time_zone.1 }}</option>\n";
{% endfor %}
result += "</select><br />\n";
result += "Observes daylight saving time: ";
result += "<input type='checkbox' ";
result += "name='observes_daylight_saving_time' ";
result += "id='observes_daylight_saving_time' ";
result += "onchange="
    'PHOTO_DIRECTORY.update_autocomplete("';
result += '"Entity_observes_daylight_saving_time_';
result += id;
result += '", "observes_daylight_saving_time");';
result += "'";
if (entity.fields.observes_daylight_saving_time)
{
    result += " checked='checked'";
}
result += " /></p>";
if (entity.fields.time_zone)
{
    result += "<p>Local time:";
    result += "<span id='local_time_zone'>";
    result += entity.fields.time_zone;
    result += "</span>";
    result += "<span id='local_time'></span></p>";
}
result += "<p>Department: <strong>";
```

The code fails to select the Entity's time zone; we remedy that after `build_profile()` is called:

```
PHOTO_DIRECTORY.load_profile = function(id)
{
    PHOTO_DIRECTORY.current_profile = id;
    if (PHOTO_DIRECTORY.SHOULD_TURN_ON_HIJAXING)
    {
        try
        {
```

```

$("#profile").html(PHOTO_DIRECTORY.build_profile(id));
if (PHOTO_DIRECTORY.entities[id].fields.time_zone !=
    null)
{
    $("#time_zone").val(
        PHOTO_DIRECTORY.entities[id].fields.time_zone);
}
PHOTO_DIRECTORY.register_update();

```

We add a bit of styling:

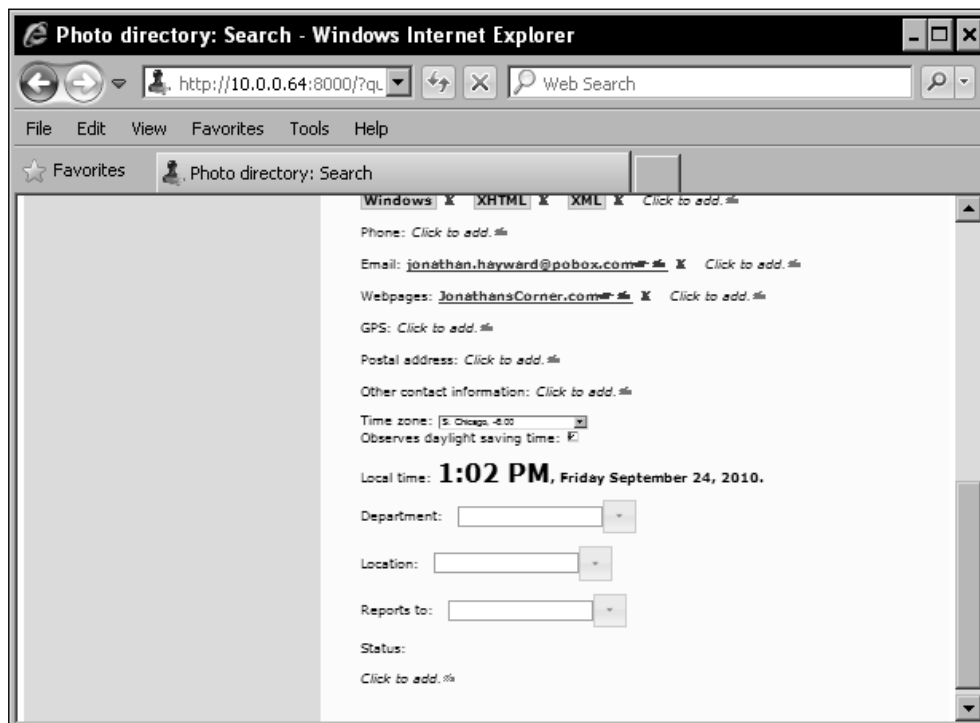
```

#local_time
{
    font-weight: bold;
}

#local_time strong
{
    font-size: 2em;
}

```

And with that, we have a clear local time slot in our profile:



Summary

We have covered some tinkering and tweaks, and bugfixes along the way. Our directory has fewer rough spots! These include: updating the code to handle deletion correctly after we have added undoing and we retain some "deleted" items in the database, tweaking a standard plugin to enhance its behavior, improving a basic widget in terms of its usability, and adding a good first pass at a world time zone "What's their local time?" slot on the profile.

Let us continue to take a look at usability itself in the next chapter.

Where to buy this book

You can buy Django JavaScript Integration: AJAX and jQuery from the Packt

Publishing website: <https://www.packtpub.com/django-javascript-integration-ajax-and-jquery/book>

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



www.PacktPub.com

For More Information:

www.packtpub.com/django-javascript-integration-ajax-and-jquery/book