

# Best-practice solutions for common Ajax use-cases with **Prototype**

Christophe Porteneuve  
CTO, Ciblo + Prototype Core member



# Hi

- Christophe, 30, French, Paris-based
- Day job = mostly Rails + Prototype/Scripty
- Prototype Core since January 2007
- The Bungee Book

# So what's this about?

- Prototype's one of the most popular JS libraries out there
- It's know a few "growth spurts," but people haven't always taken the time to catch up
- A lot of sub-par / suboptimal code out there, that should be rethought
- A few well-honed patterns will get you *far*.

# Code-heavy, demo-rich

- We're going to dive into code
- The slides feature the key points, the concepts, the workflow, the code highlights
- The side demos provide the concrete examples and working implementations
- I'll post both online

# On the server side...

- I wish I went Rails...
- ...but not all of you may know it well
  - ▶ And to be honest, it's not *necessary*.
- So we'll go self-contained PHP on this one
- Hey, a silex never killed anyone
  - ▶ (Yeah, troll inside)

# Ninjahood optional

- OK, you do need **JavaScript 101**
- And I **won't** explain what Ajax/XHR **is**
- And I'll assume you know what **progressive enhancement** means
- Hey, this is **@mediaAjax**.
- Still, you won't need strong JS-fu, you won't need to be an actual JS ninja.

# Checkbox lists

# (un)check. that's it.

- Requiring manual submission of updates in a checkbox list is a pain
  - ▶ If the list's long, it forces you to scroll down, submit, reload, then perhaps even rescroll to get back where you were.
  - ▶ You want to be able to keep the list there, no scrolling, no submitting. Yet the server should reflect any (un)checking you do.

# Use case examples

- has-many relationship between the page's main entity and other items, in a backoffice
  - ▶ Products in the current category/shelf/aisle
  - ▶ E-mail push types a customer subscribes to
- To-Do lists, *a la* 37signals
- Enabled/disabled status of items in a set
  - ▶ Active images/banners for the homepage rotation

# A checklist example

- What things bring Jake the least amount of pain?
- In ordinary times, I'd go REST, you betcha.
- ...but I said I used raw PHP!
- So I won't PUT to `/people/jake`, alright? Just a POST to some session-persisting URL.

# The markup

```
<form id="checklist" method="post" action="painless.php">
  <p>
    <input type="checkbox" name="painless_ids[]" value="0" id="chkProduct_0" />
    <label for="chkProduct_0">Kitkat chunky</label>
  </p>
  <p>
    <input type="checkbox" name="painless_ids[]" value="1" id="chkProduct_1" />
    <label for="chkProduct_1">Canadian ham flavour seabrook crips</label>
  </p>
  <p>
    <input type="checkbox" name="painless_ids[]" value="2" id="chkProduct_2" />
    <label for="chkProduct_2">Battlestar Galactica</label>
  </p>
  ...
  <input type="submit" value="Update" />
</form>
```

# The server side

- OK, let's keep it trivial, no DB, just a session

```
<?php
$_SESSION['painless_ids'] = $_POST['painless_ids'];

usleep(rand(1, 10) * 100000); // Simulate server load

if ($_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest') {
    header('HTTP/1.1 200 Painless items saved.');
```

```
} else {
    header('Location: index.php');
}
?>
```

# The JavaScript #1 aka “Copy/paste montage fest”

```
document.observe('dom:loaded', function() {
  $$('#checklist input[type="checkbox"]').each(function(cb) {
    cb.observe('click', function(e) {
      new Ajax.Request('painless.php', {
        parameters: $('checklist').serialize(),
        onCreate: function() { $('spinner').show(); },
        onComplete: function() { $('spinner').hide(); }
      });
    });
  });
  $('checklist').down('input[type="submit"]').hide();
});
```

# The JavaScript #1 aka “Copy/paste montage fest”

```
document.observe('dom:ready', function() {  
  $$('#checkboxlist input[type="checkbox"]').each(function(cb) {  
    cb.observe('click', function() {  
      new Ajax.Request('/ajax/checkbox', {  
        parameters: $$('#checkboxlist').serialize(),  
        onCreate: function() { $('#spinner').show(); },  
        onComplete: function() { $('#spinner').hide(); }  
      });  
    });  
  });  
  $('#checkboxlist').down('input[type="submit"]').hide();  
});
```



# The JavaScript #1 aka “Copy/paste montage fest”

```
document.observe('dom:loaded', function() {  
  $('#checklist input[type="checkbox"]') each(function(cb) {  
    cb.observe('click', function(e) {  
      new Ajax.Request('painless.php', {  
        parameters: $('checklist').serialize(),  
        onCreate: function() { $('spinner').show(); },  
        onComplete: function() { $('spinner').hide(); }  
      });  
    });  
  });  
  $('checklist').down('input[type="submit"]').hide();  
});
```

This CSS selection is not reusable outside this particular checklist... Prefer `$('checklist').select(...)`

# The JavaScript #1 aka “Copy/paste montage fest”

```
document.observe('dom:loaded', function() {  
  $$('#checklist input[type="checkbox"]').each(function(cb) {  
    cb.observe('click', function(e) {  
      new Ajax.Request('painless.php', {  
        parameters: $('checklist').serialize(),  
        onCreate: function() { $('spinner').show(); },  
        onComplete: function() { $('spinner').hide(); }  
      });  
    });  
  });  
  $('checklist').down('input[type="submit"]').hide();  
});
```

Our iterator code consists of a single method call (observe), therefore we should use invoke.

# The JavaScript #2

## aka “It’s just shorter”

```
document.observe('dom:loaded', function() {
  $('checklist').select('input[type="checkbox"]').invoke('observe', 'click',
    function(e) {
      new Ajax.Request('painless.php', {
        parameters: $('checklist').serialize(),
        onCreate: function() { $('spinner').show(); },
        onComplete: function() { $('spinner').hide(); }
      });
    });
  $('checklist').down('input[type="submit"]').hide();
});
```

**Not DRY: the proper target is already in our markup (action=). Also, what if the form goes GET?**

# The JavaScript #2

## aka “It’s just shorter”

```
document.observe('dom:loaded', function() {
  $('checkboxlist').select('input[type="checkbox"]').invoke('observe', 'click',
    function(e) {
      new Ajax.Request('painless.php', {
        parameters: $('checkboxlist').serialize(),
        onCreate: function() { $('spinner').show(); },
        onComplete: function() { $('spinner').hide(); }
      });
    });
  $('checkboxlist').down('input[type="submit"]').hide();
});
```

We’re Ajaxifying the form, so obviously we’re going to use its serialization. This is such a common pattern...

# The JavaScript #3

## aka “It’s just DRYer”

```
document.observe('dom:loaded', function() {
  $('checkboxlist').select('input[type="checkbox"]').invoke('observe', 'click',
    function(e) {
      $('checkboxlist').request({
        onCreate: function() { $('spinner').show(); },
        onComplete: function() { $('spinner').hide(); }
      });
    });
  $('checkboxlist').down('input[type="submit"]').hide();
});
```

**Bloated:** I define a handler for *each friggin' checkbox*.  
What if I add more boxes later? What if there are tons?

# Event delegation, baby!

(or Chris Heilmann will hunt you down)

- Why listen to an event on every item, when you can listen once on the **container**?
- Adding/removing items is a breeze
- However, you need to **check the source of your event**, to only handle the elements you're interested in and leave the rest alone
- Gotcha: only works for **bubbling** events.

# The JavaScript #4 aka “We’re getting there”

```
document.observe('dom:loaded', function() {
  $('checklist').observe('click', function(e) {
    if (!e.findElement('input[type="checkbox"]'))
      return;
    this.request({
      onCreate: function() { $('spinner').show(); },
      onComplete: function() { $('spinner').hide(); }
    });
  });
  $('checklist').down('input[type="submit"]').hide();
});
```

Always use `findElement` (markup evolution-resistant) instead of `element()`. Also notice the default binding.

# The JavaScript #5

## aka “Stop inlining everything”

```
function sendCheckboxes(e) {
  if (!e.findElement('input[type="checkbox"]'))
    return;
  this.request({
    onCreate: function() { $('spinner').show(); },
    onComplete: function() { $('spinner').hide(); }
  });
}

document.observe('dom:loaded', function() {
  $('checklist').observe('click', sendCheckboxes);
  $('checklist').down('input[type="submit"]').hide();
});
```

Putting the handler in its own little function makes for a cleaner page initialization.

# The JavaScript #6

## aka “Chaining isn’t just for jQuery”

```
function sendCheckboxes(e) {
  if (!e.findElement('input[type="checkbox"]'))
    return;
  this.request({
    onCreate: function() { $('spinner').show(); },
    onComplete: function() { $('spinner').hide(); }
  });
}

document.observe('dom:loaded', function() {
  $('checklist').observe('click', sendCheckboxes).
    down('input[type="submit"]').hide();
});
```

**Beware of the chaining order: using down before observe has totally different semantics!**

# The JavaScript #7

aka “One spinner to rule them all”

```
Ajax.Responders.register({
  onCreate: function() { $('spinner').show(); },
  onComplete: function() {
    if (0 == Ajax.activeRequestCount)
      $('spinner').hide();
  }
});

function sendCheckboxes(e) {
  if (!e.findElement('input[type="checkbox"]'))
    return;
  this.request();
}

document.observe('dom:loaded', function() {
  $('checklist').observe('click', sendCheckboxes).
    down('input[type="submit"]').hide();
});
```

Factoring out the spinner = concise business code!

# The JavaScript #8

(just one tiny last bit)

```
Ajax.Responders.register({
  onCreate: Element.show.curry('spinner'),
  onComplete: function() {
    if (0 == Ajax.activeRequestCount)
      $('spinner').hide();
  }
});
```

Just calling an extended method on an element?  
Go curry on the namespaced version!

```
Ajax.Responders.register({
  onCreate: Element.show.curry('spinner'),
  onComplete: function() {
    if (0 == Ajax.activeRequestCount)
      $('spinner').hide();
  });
});
```

```
function sendCheckboxes(e) {
  if (!e.findElement('input[type="checkbox"]'))
    return;
  this.request();
}
```

```
document.observe('dom:loaded', function() {
  $('checklist').observe('click', sendCheckboxes).
    down('input[type="submit"]').hide();
});
```

# Takeaway points

- Event delegation reigns supreme
- So does progressive enhancement
- Know the API: methods like `curry`, `request`, etc. just save you time and code
- Chaining is useful when you're repeating a long-winded subject before your methods
- Trying to put everything into one long-winded method just sucks

Live login validation

# Yeah, you're not the only one. Bummer.

- So you like this service enough that you're actually subscribing.
- And it doesn't use emails as logins, so you need to pick one.
- Hey, one more chance to get über-cool with "ubersexycool dude," yay!
- You fill it all out, submit... bang! There's another tragic case of self-delusion.

# Not just logins, either.

- Subdomains
- Trademarks
- Licence plates
- Project repositories
- E-mail addresses
- ...

# The markup

```
<form id="signUpForm" method="post" action="login.php">
  <p>
    <label for="edtReqLogin" accesskey="L">Login</label>
    <input type="text" name="login" id="edtReqLogin" tabindex="1" />
    <span id="spinner" style="display: none;"></span>
    <span id="loginNotice" style="display: none;"></span>
  </p>

  <p><input type="submit" value="Sign up!" accesskey="S" tabindex="2" /></p>
</form>
```

# The server side

```
<?php
$xmlhr = 'XMLHttpRequest' == $_SERVER['HTTP_X_REQUESTED_WITH'];
$LOGINS = array('dhh', 'dion', 'foobar', 'joe', 'jsmith', 'ubersexycooldude');
if (in_array($_POST['login'], $LOGINS)) {
    if ($xmlhr) {
        header('HTTP/1.1 409 Conflict');
        echo 'This login is not available.';
    } else {
        $_SESSION['errorMsg'] = 'This login is not available.';
        header('Location: index.php');
    }
} else {
    if ($xmlhr) {
        header('HTTP/1.1 202 Accepted');
        echo 'This login is available.';
    } else {
        header('Location: logged_in.php'); // And some actual signup/login code..
    }
}
?>
```

# The JavaScript #1 aka “Just the facts”

```
function watchLogin() {  
  new Field.Observer('edtReqLogin', 0.8, function(field, value) {  
    $('loginNotice').hide();  
    new Ajax.Updater('loginNotice', 'login.php', {  
      parameters: { login: value },  
      onComplete: Element.show.curry('loginNotice')  
    });  
  });  
}  
  
document.observe('dom:loaded', watchLogin);
```

It's pretty good already (Field.Observer, curry calls), but it lacks a lot of polish, starting with color coding...

# The JavaScript #2

## aka “Color coded”

```
...
var updater = new Ajax.Updater('loginNotice', 'login.php', {
  parameters: { login: value },
  onComplete: function() {
    if (updater.success()) {
      field.removeClassName('taken');
      $('loginNotice').removeClassName('failure').show();
    } else {
      field.addClassName('taken');
      $('loginNotice').addClassName('failure').show();
    }
  }
});
...
```

There's a lot of repetition in this code. The code on the success/failure branching is very, very close.

# The JavaScript #3 aka “Method selector”

```
...
var updater = new Ajax.Updater('loginNotice', 'login.php', {
  parameters: { login: value },
  onComplete: function() {
    field[(updater.success() ? 'remove' : 'add') + 'ClassName']('taken');
    $('loginNotice')[(updater.success() ? 'remove' : 'add') + 'ClassName']('failure').show();
  }
});
...
```

Remember, `obj[x]` is property selection based on expression “x”. Now how about reducing lookups?

# The JavaScript #4

## aka “Just look once”

```
function watchLogin() {
  var loginNotice = $('#loginNotice');
  new Field.Observer('edtReqLogin', 0.8, function(field, value) {
    loginNotice.hide();
    var updater = new Ajax.Updater(loginNotice, 'login.php', {
      parameters: { login: value },
      onComplete: function() {
        field[(updater.success() ? 'remove' : 'add') + 'ClassName']('taken');
        loginNotice[(updater.success() ? 'remove' : 'add') + 'ClassName']('failure').show();
      }
    });
  });
}
```

Finally, let's prevent the user from submitting the form while we're checking availability (and if it's taken).

# The JavaScript #5

## aka “Wait a second”

```
function watchLogin() {
  var loginNotice = $('#loginNotice');
  var submit = $('#signUpForm').down('input[type="submit"]');
  new Field.Observer('edtReqLogin', 0.8, function(field, value) {
    loginNotice.hide();
    submit.disable();
    var updater = new Ajax.Updater(loginNotice, 'login.php', {
      parameters: { login: value },
      onComplete: function() {
        field[(updater.success() ? 'remove' : 'add') + 'ClassName']('taken');
        loginNotice[(updater.success() ? 'remove' : 'add') + 'ClassName']('failure').show();
      },
      onSuccess: Field.enable.curry(submit)
    });
  });
}
```

# Takeaway points

- `Ajax.Updater` is a nice shortcut
- Method/property selection through the native `[]` operator is a way underused feature of the language
  - ▶ Long-winded ternaries are sometimes worse than their if/else counterparts, though. YMMV.
- Disable parts of your UI that are not applicable / dangerous at present.

Server-side progress




# This could take a while

- File uploading
- Mail with PDF attachment (e.g. bill)
- Certificate generation
- System configuration (DNS, mailboxes...)
- Customer base export (with orders?)
- ...

# So what's the idea?

- Well, the solutions are aplenty
- Unless you're going Comet / WebSockets, you need to periodically poll the server
  - ▶ Perhaps with a "job key"
  - ▶ You'll get back a progress status
  - ▶ When it's done, you move on.

# Our example

- Simulating a long-running task on the server
- Choose a data format:
  - ▶ XHTML fragment for the progress bar 
  - ▶ JSON or plain text: completion percentage 
  - ▶ JS: tweaking the existing progress bar 
- It's really your call (indeed)

# The markup

```
<form id="processForm" method="get" action="progress.php">  
  <p><input type="submit" id="btnStart" value="Start querying" /></p>  
</form>
```

```
<p id="progress">  
  <span class="bar"></span>  
  <span class="percent">0%</span>  
</p>
```

# The server side (1/2)

```
<?php
$xmlr = 'XMLHttpRequest' == $_SERVER['HTTP_X_REQUESTED_WITH'];
$progress = isset($_SESSION['progress']) ? $_SESSION
['progress'] : 0;
if ($progress == 0 && $xmlr)
    $removeOver50 = TRUE;
if ($progress < 100) {
    $_SESSION['progress'] = min($progress + rand(5, 15), 100);
    $shotOver50 = $progress <= 50 && $_SESSION['progress'] > 50;
    $progress = $_SESSION['progress'];
}
if (!$xmlr) {
    header('Location: index.php');
    exit(0);
}
...
```

# The server side (2/2)

```
header('Content-Type: text/javascript');
if ($progress >= 100) {
    unset($_SESSION['progress']);
    echo 'gPoller.stop();';
    echo "$('btnStart').enable()";
}
if ($removeOver50)
    echo "$('progress').removeClassName('over50')";
echo "$('progress').down('.bar').setStyle('width: $progress%')";
echo "$('progress').down('.percent').update('$progress%')";
if ($shotOver50)
    echo "$('progress').addClassName('over50')";
?>
```

# The JavaScript

```
var gPoller;

function watchProgress(e) {
    e.stopPropagation();
    $('btnStart').disable();
    this.request();
    gPoller = new PeriodicalExecuter(this.request.bind(this), 1);
}

document.observe('dom:loaded', function() {
    $('processForm').observe('submit', watchProgress);
});
```

# Takeaway points

- You should consider carefully what content type makes the most sense for any given Ajax result.
- XHTML, JSON, JS... all have their pros and cons, not just format-wise but regarding (de)coupling between your client and your server sides.

Reordering' by draggin'

# Just shuffling around...

- OK, it's just everywhere
- Bookmarks
- To-Do Lists
- Photos in a photoset
- Articles in a FAQ or newsletter
- Say you want to save the order **live**

# Principles

- We'll use `script.aculo.us` for its `Sortable` class.
- Basically, we serialize the ordering as an array of IDs, and send it to the server
- RESTfully, we'd PUT on the parent resource
- Here I'll just POST to a session-persisting page, like I did for checklists.

# Key things to know

- To serialize a sortable list, all the items **must have IDs**
- These IDs need to follow a syntax pattern, by default **anyPrefix\_value**.
- The list's ID is used as param name by default, with an added **[]** suffix.

# The markup

```
<ol id="tasks">
  <li id="task_1">
    
    <input type="checkbox" name="done_ids[]" id="chkDone1" value="1" />
    <label for="chkDone1">Blah blah blah...</label>
  </li>
  <li id="task_2">
    
    <input type="checkbox" name="done_ids[]" id="chkDone2" value="2" />
    <label for="chkDone2">Lorem ipsum...</label>
  </li>
  ...
</ol>
```

# The server side

```
<?php
$_SESSION['task_ids'] = $_POST['tasks'];
header('HTTP/1.1 200 Task order saved. ');
?>
```

*“Man, I can’t believe I paid good money to see this kind of code on stage. What a rip-off!”*

# OK, the real deal then!

- If you don't use Rails, well, you asked for it!

```
def update
  @todo = Todo.find(params[:id])
  @todo.task_ids = params[:tasks]
  respond_to do |format|
    format.js { head :success }
    format.html { redirect_to to_do_path(@todo) }
  end
end
```

- There.
  - ▶ Yeah, I couldn't resist. Bad, bad Christophe.

# The JavaScript

```
function initReordering() {  
    Sortable.create('tasks', { handle: 'handle',  
        onUpdate: function(list) {  
            new Ajax.Request('reorder.php', {  
                parameters: Sortable.serialize(list)  
            })  
        }  
    });  
}  
  
document.observe('dom:loaded', initReordering);
```

Hijaxed deletion

# Peeling stuff off

- You're just removing items from a list.
- The worst thing would be to reload the whole darn page every time!
- So we go Ajax then fade it out
  - ▶ We can even strip it from the DOM once faded
- Let's keep accessible and safe, though.

# The markup

## item list

```
<ol id="tasks">
  <li id="task_1">
    <input type="checkbox" name="done_ids[]" id="chkDone1" value="1" />
    <label for="chkDone1">Blah blah blah..</label>
    <a href="delete.php?id=1" title="Delete this task" class="delete">
      ↪ </a>
  </li>
  <li id="task_2">
    <input type="checkbox" name="done_ids[]" id="chkDone2" value="2" />
    <label for="chkDone2">Blah blah blah..</label>
    <a href="delete.php?id=2" title="Delete this task" class="delete">
      ↪ </a>
  </li>
  ...
</ol>
```

# The markup

## confirm deletion

```
<h1>Delete this task?</h1>
```

```
<form id="tasksForm" method="post" action="delete.php">
```

```
  <p>Click the button below to confirm you wish to delete the task  
  "Blah blah blah."</p>
```

```
  <p>
```

```
    <input type="hidden" name="id" value="1" />
```

```
    <input type="submit" value="Yes, delete!" />
```

```
  </p>
```

```
</form>
```

# The server side

```
$xhr = 'XMLHttpRequest' == $_SERVER['HTTP_X_REQUESTED_WITH'];  
$id = $_REQUEST['id'];  
if ('POST' == $_SERVER['REQUEST_METHOD']) {  
    $_SESSION['task_ids'] = array_diff($_SESSION['task_ids'], array($id));  
    if ($xhr)  
        header('HTTP/1.1 200 OK (Task deleted.)');  
    else  
        header('Location: index.php');  
    exit(0);  
}  
// Then the deletion confirmation markup...
```

# The JavaScript

```
function handleDeleters(e) {
  var deleter = e.findElement('a.delete');
  if (!deleter)
    return;
  deleter.blur();
  e.stopPropagation();
  var task = deleter.previous('label').innerHTML.unescapeHTML();
  if (!confirm("Delete task " + task + "?"))
    return;
  new Ajax.Request(deleter.href, {
    method: 'delete',
    onSuccess: function() { deleter.up('li').fade({ duration: 0.5 }); },
    onFailure: function() { alert("Deletion failed!"); }
  });
}

document.observe('dom:loaded', function() {
  $('tasks').observe('click', handleDeleters);
});
```



Questions?

Shoot!