FULL SAIL
UNIVERSITY

programming for web applications

12

jQuery.Forms.Controls.Dialogs

jQuery.Forms

jQuery.Forms

Things we can do with Forms

- Guest Books
- Feedback mechanism
- Fancy Form Validation Messages
- Cool Dialog boxes
- Submit Event
- Validation
- Validation Plugins
- Form Hints
- Checkboxes and more...

Client-side Form Validation

- Client-side form validation with jQuery should only be used to assist your user
- Should never be relied upon to prevent certain types of data being sent to the server
- Users with JavaScript disabled will be unhindered by your jQuery validation, so they can submit any values they want
- If there's any security risk from users submitting malicious data security issues your forms, that data needs to be thoroughly validated on the server side.
- Open Web Application Security Project is a great place to start if you need some pointers on securing your applications

jQuery.Forms

querySelectorAll (qSA)

- Makes elements from forms far more efficient. (Accessing setting form values)
- Highly responsive for compliant browsers
- Provide jQuery with a selector that will invoke it...

```
$('input[type=text]').css('background-color', 'pink')
```

- Instead of using **:input** or **:text** in a filter, we use querySelectorAll selector. (qSA)
- The **:password** filter method, for example, selects all elements that are password inputs from all the inputs in the page, and is reasonably quick at 13,589 operations per second.
- If you preferred it over **input[type='password']** , though, you' d be losing out on nearly another 13,000 operations.

jQuery.Forms

querySelectorAll (qSA)

When tested on Chrome

Type Password	$(“[type=‘password’]”);	20% slower
Filter Password	$(“:password”);	91% slower
Input Filter Password	$(“input:password”);	56% slower
Filter Method Password	$(“input”).filter(“:password”);	49% slower
Input Type Password	$(“input[type=‘password’]”)	Fastest Method

jQuery.Forms

Radio, Image and Submit

- [type='radio'] , [type='image'] , or [type='submit']
- Rumor has it that filters are slated to be deprecated, so use them at your own risk.
- val function returns the value of a form field
- :checked and :selected (helpful when using a radio and checkbox buttons

```
if($(this).is(':checked'))
```

Radio, Image and Submit

- Let' s test to see if any text boxes in a form are empty

```
$('#myform').bind('submit', function(event) {
  $('[type=text]').each(function() {
    if(!$(this).val().length) {
      event.preventDefault();
      $(this).css('border', '2px solid red');
    }
  });
});
```


Radio, Image and Submit

- Val action works for select boxes and radio buttons too.

```
$('[type=input]').change(function(){
  alert($(this).val());
});

$('[name=yes]').bind('click', function(){
  alert($(this).val());
});
```

Radio, Image and Submit

- Testing for empty fields when the user moves to the next field using the blur event. (inline validation)

```
$(':input').blur(function() {
  if ($(this).val().length == 0) {
    $(this)
      .addClass('error')
      .after('<span class="error">This field must … </span>');
  }
});
$(':input').focus(function() {
  $(this)
    .removeClass('error')
```

Avoid Over-validating!

- Keep it simple! Offer hints, sample inputs, and guidance
- To enable simple validation in HTML5, you only need to add required as an attribute in a compliant browser:

```
<input value="Crystal" placeholder="Web Developer" required>
```

Avoid Over-validating!

jQuery.Forms

Submit Event

- A better technique than listening for a click event on the submit button
- It will also fire if the user submits the form by pressing the Enter key.
- Use event.preventDefault to ensure that the form will only be submitted once we' re satisfied we have the required details in our form.
- Check all the text boxes in the form. If any are left empty, we' ll pop up a message and focus on the offending element:

```
$("form").submit(function(event){
  var error = false;
  $(this).find("[type=text]").each(function(){
    if (!$(this).val().length) {
      alert("Textboxes must have a value!");
      $(this).focus();
      error = true;
      return false; // Only exits the “each” loop
    }
});
if (error) {
  event.preventDefault();
}
});
```

Generalize your Validation

- If you plan your forms carefully and develop a consistent naming standard, you can use jQuery to generalize your validation so that it can apply to many forms.
- If you need really bulletproof validation and would rather spend your time designing the user interaction, consider the Validation Plugin.

Validation Plugin - Inline Validation

- Can add sophisticated and customizable inline validation to most forms with minimal effort.
- Sign-up form that includes password and password confirmation fields:

```
<div id="signup">
<h2>Sign up</h2>
<form action="">
<div>
<label for="name">Name:</label>
<input name="name" id="name" type="text"/>
</div>
<div>
<label for="email">Email:</label>
<input name="email" id="email" type="text"/>
</div>
<div>
<label for="website">Web site URL:</label>
<input name="website" id="website" type="text" />
</div>
<div>
<label for="password">Password:</label>
<input name="password" id="password" type="password" />
</div>
<div>
<label for="passconf">Confirm Password:</label>
<input name="passconf" id="passconf" type="password" />
</div>
<input type="submit" value="Submit!" />
</form>
</div>
```

Validation Plugin - Call Validate

- Sign-up form that includes password and password confirmation fields
- to use the Validation Plugin, call validate on a selection of our form, passing it any options we want to use.
- The most important option is rules
- You need to define rules used to validate the users' input:
- A considerable number of predefined validation rules available, and you can define your own.
- Some examples: required , email , url , minlength , and equalTo .
- Inside the rules object, we define an object for each form field, using the field' s name attribute.
- equalTo allows us to specify a jQuery selector pointing at another form field, the contents of which will be checked against the current field to see if they' re the same.

Call Validate

- The Validation plugin will add a new label element after each form field to contain the error message; by default, this will have a class of error , so you' re free to style it in as you' d like.
- The plugin will only display a message if a field' s value is invalid.
- User research has shown that users complete forms more quickly and confidently if they' re also provided with feedback for correct entries.
- This is a powerful plugin with the sort of features that we hope to see in the browser itself one day.

Maximum Length Indicator

- Setting a limit on the length of input
- Ex: Twitter
- Displaying the remaining characters next to the form field give users have clear expectations of how much they can type.
- We' ll set a class of maxlength on the textarea we want to target with this effect.
- After we append the span, the textarea is still the selected element. We want to modify the new span , so we move to it with the next action.
- Hide the span , but now we need to go back to our form element to add an event handler, so we use the end action.
- When we call end , the selection moves back to the state it was in before we called next .

```
$('.maxlength')
  .after("<span></span>")
  .next()
  .hide()
  .end()
  .keypress(function(event) {
// handle key presses;
});
```

Maximum Length Indicator

- Back on the form element, we attach a keypress event handler
- This event fires whenever a key is pressed
- We grab the value of the element and use the JavaScript length property to give us its length.
- If the current number of characters is greater than the maximum length, we' ll prevent the key press from registering by using the preventDefault action of the event.

```
var current = $(this).val().length;
if (current >= 130) {
if (event.which != 0 && event.which !== 8) {
event.preventDefault();
}
}
```

Maximum Length Indicator

- When handling a keypress event, the event has a which property corresponding to the ASCII code of the key pressed.
- Note that we' ve allowed the delete (ASCII code0) and backspace (ASCII code 8) keys to function regardless of the number of characters. If we didn' t do this, the user could paste in a response that exceeded the limit, yet be unable to delete any characters to make it fit:
- Display the number of remaining characters in the span
- Move to the next element, make sure it' s visible, and display the results of our simple calculation to indicate how many more characters are allowed.
- Add a method to change the span' s style, making it even easier to know when the 140-character cap is reached.

```
$(this).next().show().text(130 - current);
```

Form Hints

- To decrease the amount of space a form takes up on the page, move the label for a form field inside the input itself.
- Users move their focus to the field, the label vanishes, allowing them to start typing.
- If they leave the field empty and move away, the original label text appears back in its place.
- This technique is only appropriate for short and simple forms.
- In larger forms, it' s too easy for users to lose track of what each particular field is for in the absence of visible labels.
- That said, for simple forms like login or search forms, where most users are very familiar with what each field is for, it can be a great way to save space and streamline your interface.
- Using the data action, we are able to use the HTML5 placeholder attribute
- Support troubles: there' s still a considerable number of browsers that don' t support it, and those that do require vendor-prefixed CSS to style it.

jQuery.Forms

Form Hints

- Store the default value in data for each clearable item, and if the value is still empty when the user leaves, we' ll restore it from there
- Go through each element and save the default value when the document loads
- Keep track of the focus and blur events that will fire whenever the user moves into or out of our input's
- On focus , we test if the value of the text box is the same as our default text; if it is, we clear the box in preparation for the user' input
- On the way out, check to see if the text box is empty, and if it is we put the original value back in.
- Add and remove a class as we go
- Allows us to style the form fields differently when they' re displaying the hint.

jQuery.Forms

Form Hints

```
$('input.clear').each(function() {
  $(this)
    .data('default', $(this).val())
    .addClass('inactive')
    .focus(function() {
      $(this).removeClass('inactive');
      if($(this).val() === $(this).data('default') ||
        ➥$(this).val() === '') {
        $(this).val('');
      }
    })
    .blur(function() {
     if($(this).val() === '') {
       $(this).addClass('inactive').val($(this).data('default'));
     }
   });
});
```

Check All Checkboxes

- With text inputs firmly under our control, it' s time to move on to other form controls.
- Each category has a statistic checkbox
- A Check all box, so that the user can toggle all the checkboxes off or on at once.
- Knowing the jQuery form filters makes this task a walk in the park. We just have to select all checkboxes in the same group, and check or clear them.
- The way we group checkboxes together in HTML forms is by giving all the related items the same name:
- The last checkbox the special class of check-all . This box will act as our master checkbox: when it is checked or unchecked, our code springs to life.
- Create a selector that looks like :checkbox[name=reason]
- Set all the related checkboxes to have the same checked value as our master checkbox.

Check All Checkboxes

```
<div class="stats">
<span class="title">Reason for Celebrity</span>
<input name="reason"
type="checkbox" value="net" />Famous on the Internet<br/>
<input name="reason"
type="checkbox" value="crim" />Committed a crime<br />
<input name="reason"
type="checkbox" value="model" />Dates a supermodel<br />
<input name="reason"
type="checkbox" value="tv" />Hosts a TV show<br />
<input name="reason"
type="checkbox" value="japan" />Big in Japan<br />
<hr />
<input class="check-all"
name="reason" type="checkbox" /><span>Check all</span>
</div>
```

```
$('.check-all:checkbox').change(function() {
var group = ':checkbox[name=' + $(this).attr('name') + ']';
$(group).attr('checked', $(this).attr('checked'));
});
```

Auto Complete

```
<label for="location">Last known whereabouts:</label>
<input type="text" id="location"/>
```

```
var cities = ['Paris', 'New York', 'Milan', 'Stockholm' … ];
$('#location').autocomplete(cities,{
autoFill: true,
selectFirst: true,
width: '240px'
});
```

```
.autocomplete(cities)
.result(function(event, selection) {
alert(selection);
});
```

```
var cities = ['Paris', 'New York', 'Milan', 'Stockholm' … ];
$('#location').autocomplete({
source: cities
});
```

jQuery.Controls

jQuery.Controls

Buttons

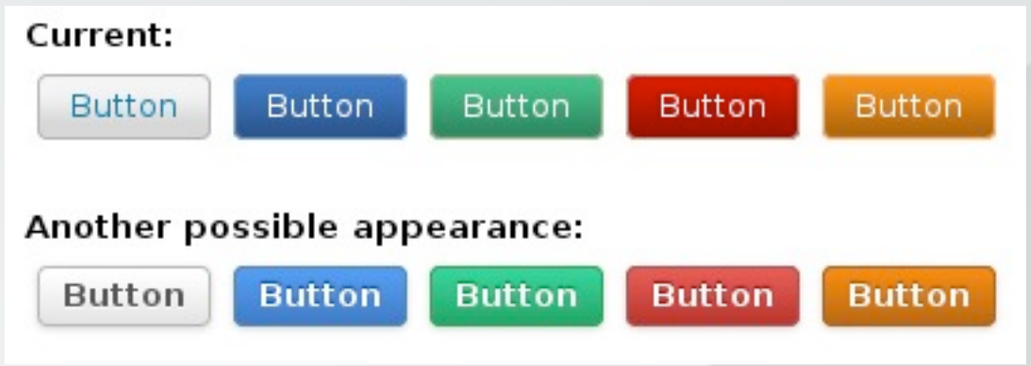
```
<fieldset id="radio">
<input type="radio" id="radio1" name="radio" /><label
➥for="radio1">Find a celebrity</label>
<input type="radio" id="radio2" name="radio"
checked="checked"
➥/><label for="radio2">Report a celebrity</label>
<input type="radio" id="radio3" name="radio" /><label
➥for="radio3">Become a celebrity</label>
</fieldset>
```

```
$('#radio').buttonset();
```

UI Buttons

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI Button - Default functionality</title>
  <link rel="stylesheet" href="//code.jquery.com/ui/1.10.4/themes/smoothness/jquery-ui.css">
  <script src="//code.jquery.com/jquery-1.9.1.js"></script>
  <script src="//code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
  <link rel="stylesheet" href="/resources/demos/style.css">
  <script>
  $(function() {
    $( "input[type=submit], a, button" )
      .button()
      .click(function( event ) {
        event.preventDefault();
      });
  });
  </script>
</head>
<body>

<button>A button element</button>

<input type="submit" value="A submit button">

<a href="#">An anchor</a>


</body>
</html>
```

UI Buttons


Current:
Button
Button
Button
Button
Button
Another possible appearance:
Button
Button
Button
Button
Button

Date Picker

```
<input type="text" id="date" />
```

```
$("#date").datepicker();
```

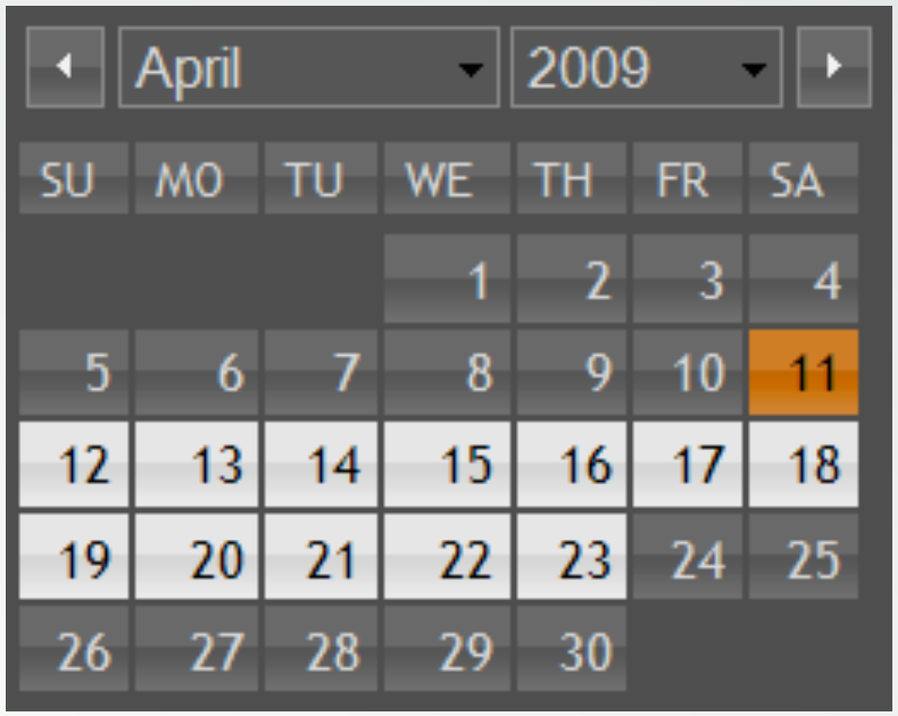
```
$('#date').datepicker({
showOn: 'both',
buttonText: 'Choose a date',
buttonImage: 'calendar.png',
buttonImageOnly: true,
numberOfMonths: 2,
maxDate: '0d',
minDate: '-1m',
showButtonPanel: true
});
```

Date Validation

You may have set a maximum date for the date picker, but users are still able to select a date outside of that range, and enter it into the text box manually. If you must ensure that dates are within a given range, you need to be performing validation on the server side. The date ranges you specify in the date picker options are to assist your users in picking valid options; that way, they avoid submitting a form that contains frustrating errors.

jQuery.Controls

Date Picker



jQuery.Controls

Sliders

```
<div id="price-range">
<form action="">
 <fieldset>
   <label for="min">Minimum Price:</label>
   <select id="min" name="min">
     <option value="0">0</option>
     <option value="10">10</option>
     <option value="20">20</option>
      ⋮
     <option value="80">80</option>
     <option value="90">90</option>
   </select>
   <br/>
   <label for="max">Maximum Price:</label>
   <select id="max" name-"max">
     <option value="10">10</option>
     <option value="20">20</option>
     <option value="30">30</option>
      ⋮
     <option value="100">100</option>
   </select>
 </fieldset>
</form>
</div>
```

```
var max = $('#max').val();
var min = $('#min').val();
var rangeSlider = $('<div></div>')
.slider({
min: 0,
max: 100,
step: 10,
values: [min, max],
range: true,
animate: true,
slide: function(e,ui) {
$(this)
.parent()
.find('#min')
.val(ui.values[0]);
$(this)
.parent()
.find('#max')
.val(ui.values[1]);
showCelebs();
}
})
.before('<h3>Drag the slider to filter by price:</h3>');
$('#price-range').after(rangeSlider).hide();
```

jQuery.Controls

Sliders

jQuery.Controls

Progress Bar

```
<form action="">
<fieldset>
<legend>StarChirp</legend>
<textarea id="chirper" rows="10" cols="50"></textarea>
<div id="console">
<div id="bar"></div>
<div id="count">0</div>
</div>
<button type="submit">Chirp!</button>
</fieldset>
</form>
```

```
$('#bar').progressbar();
```

```
$('#chirper')
.val('')
.keyup(function(e) {
var characters = 140;
var chirp = $('#chirper').val();
var count = chirp.length;
if (count <= characters) {
$('#bar').progressbar('value', (count / characters) * 100);
$('#count').text(count);
} else {
$('#chirper').val(chirp.substring(0,characters));
}
});
```

Progress Bar

dialogs.**Notifications**

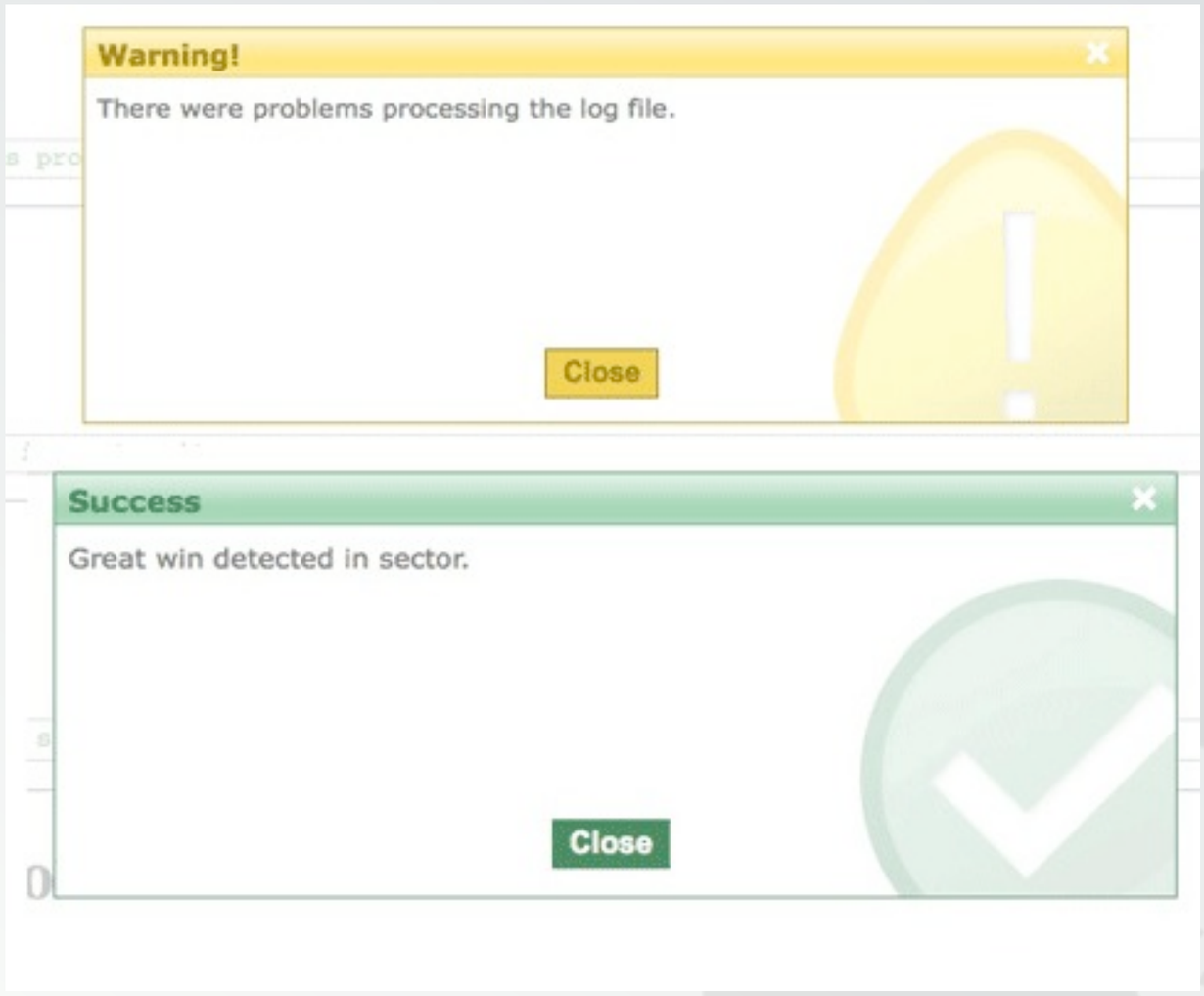
dialogs.Notifications

Simple Modal Dialog

- Ajax-enabled web applications become more complex, the breadth of information that needs to be conveyed is growing: validation messages, status updates, error handling messages, and so on.
- Modal dialogs are notifications that pop up in the user's face and must be acted on if the user wants to continue.
- People tend to dislike popups,so they should only be used if the interaction is essential.
- Example: End User License Agreement (EULA).A modal dialog looks strikingly like a lightbox with some buttons.
- We can have multiple dialogs that use the same lightbox elements

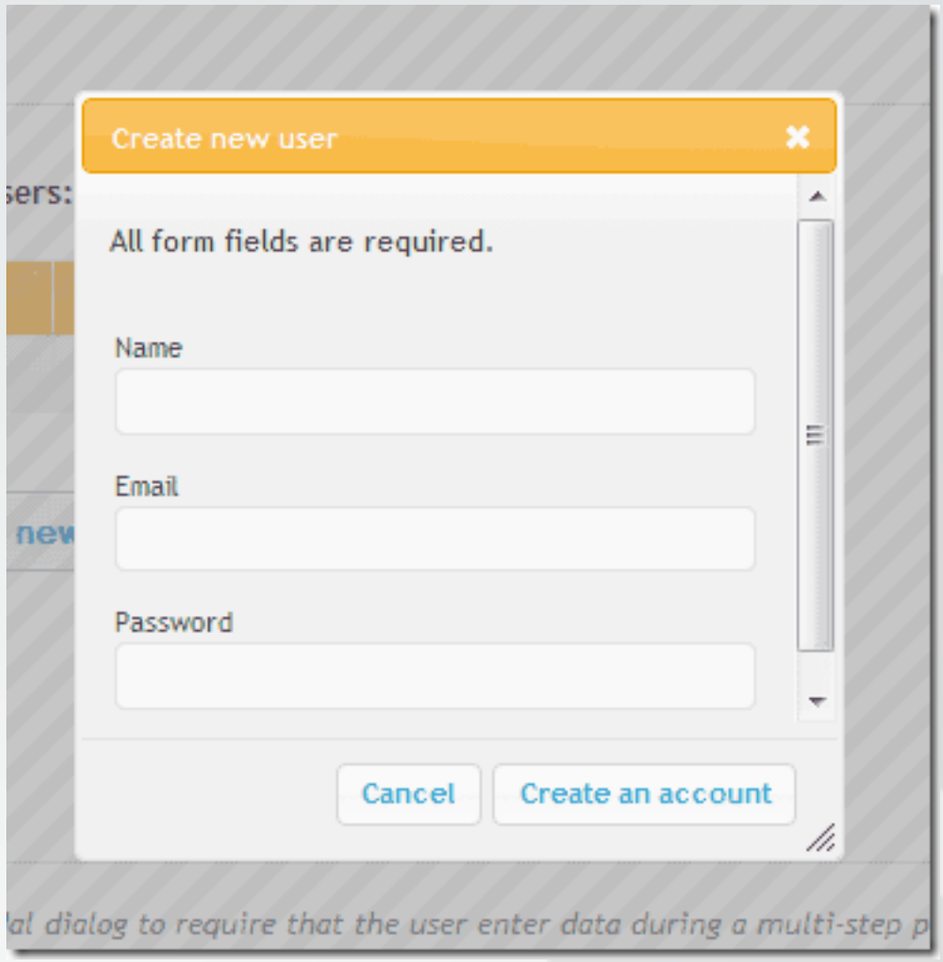
dialogs.Notifications

Simple Modal Dialog



dialogs.Notifications

jQuery UI Dialog



dialogs.Notifications

growl-style Notifications

```
<div id="growl"></div>
```

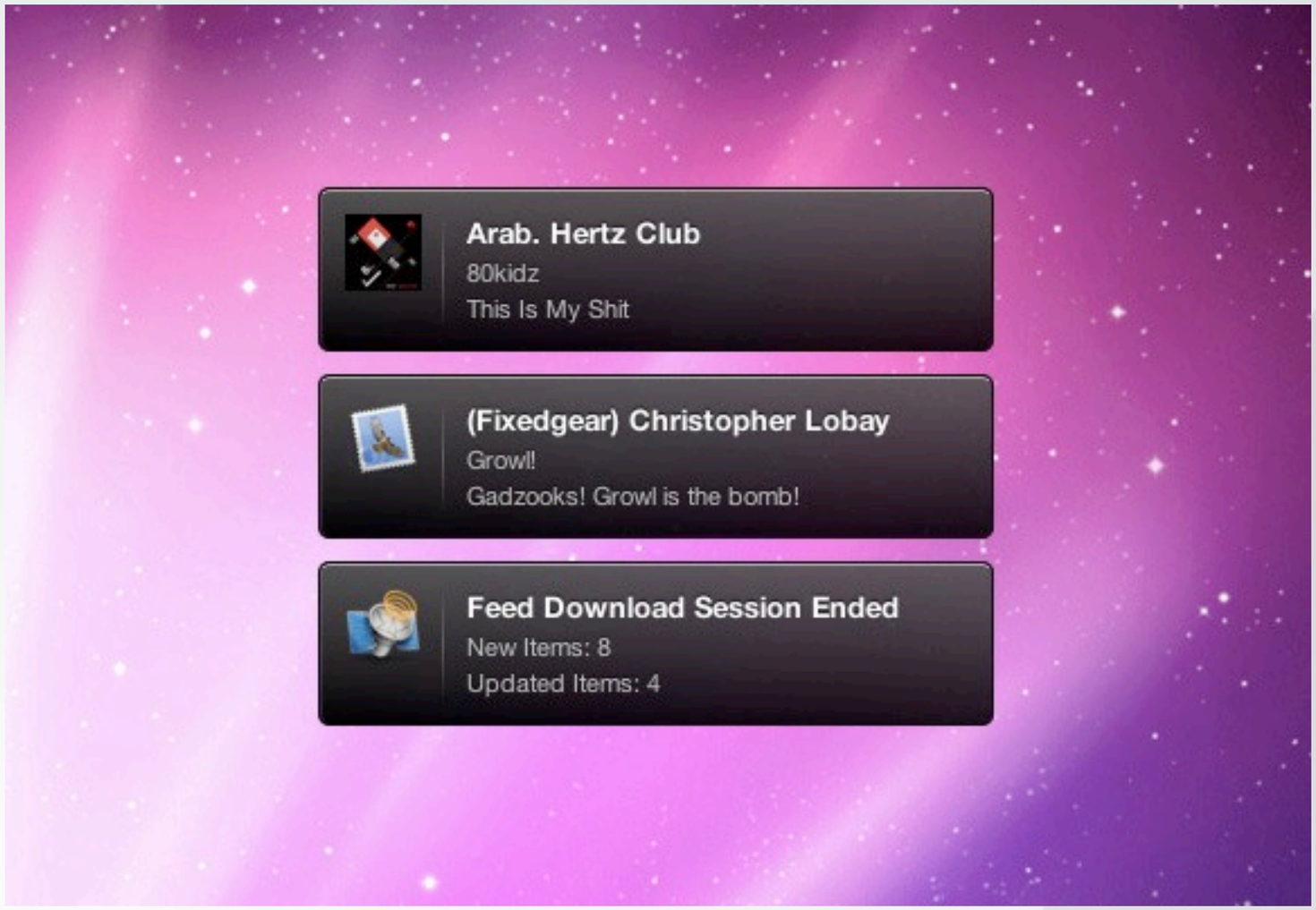
```
#growl {
position: absolute;
bottom: 0;
right: 5px;
width: 320px;
z-index: 10;
}
```

```
function addNotice(notice) {
$('<div class="notice"></div>')
.append('<div class="skin"></div>')
.append('<a href="#" class="close">close</a>')
.append($('<div class="content"></div>').html(notice))
.hide()
.appendTo('#growl')
.fadeIn(1000);
}
```

```
.notice {
position: relative;
min-height: 30px;
}
.skin {
position: absolute;
background-color: #000000;
bottom: 0;
left: 0;
opacity: 0.6;
right: 0;
top: 0;
z-index: -1;
-moz-border-radius: 5px; -webkit-border-radius: 5px;
}
.close {
background: transparent url('button-close.png') 0 0 no-repeat;
text-indent: -9999px;
position: absolute;
top: 2px;
right: 2px;
width: 26px;
height: 26px;
}
```

dialogs.Notifications

growl-style Notifications



dialogs.Notifications

1-up Notifications

- A small message (generally a single word) will appear at the point the action has taken place, then fade upwards and quickly away
- When a target (an element that has the wishlist class) is clicked, we call a custom function that sets our notification in motion. The custom function takes a reference to the current object and a callback function to run when the interaction is complete. This function will move the selection to the link (via the prev action) and set its text to Added :
- Our custom function features nothing new to us at this point: it simply moves to the hidden span element and displays it. Now the message is visible to the end user. We then kick off an animation that adjusts the span' s top and opacity properties, so as to move it upwards and fade it out simultaneously:

dialogs.Notifications

1-up Notifications

```
<a class="wishlist" href="#">Add to wishlist</a>

.adding {
position:relative;
left:-35px;
top:-4px;
display:none;
}
```

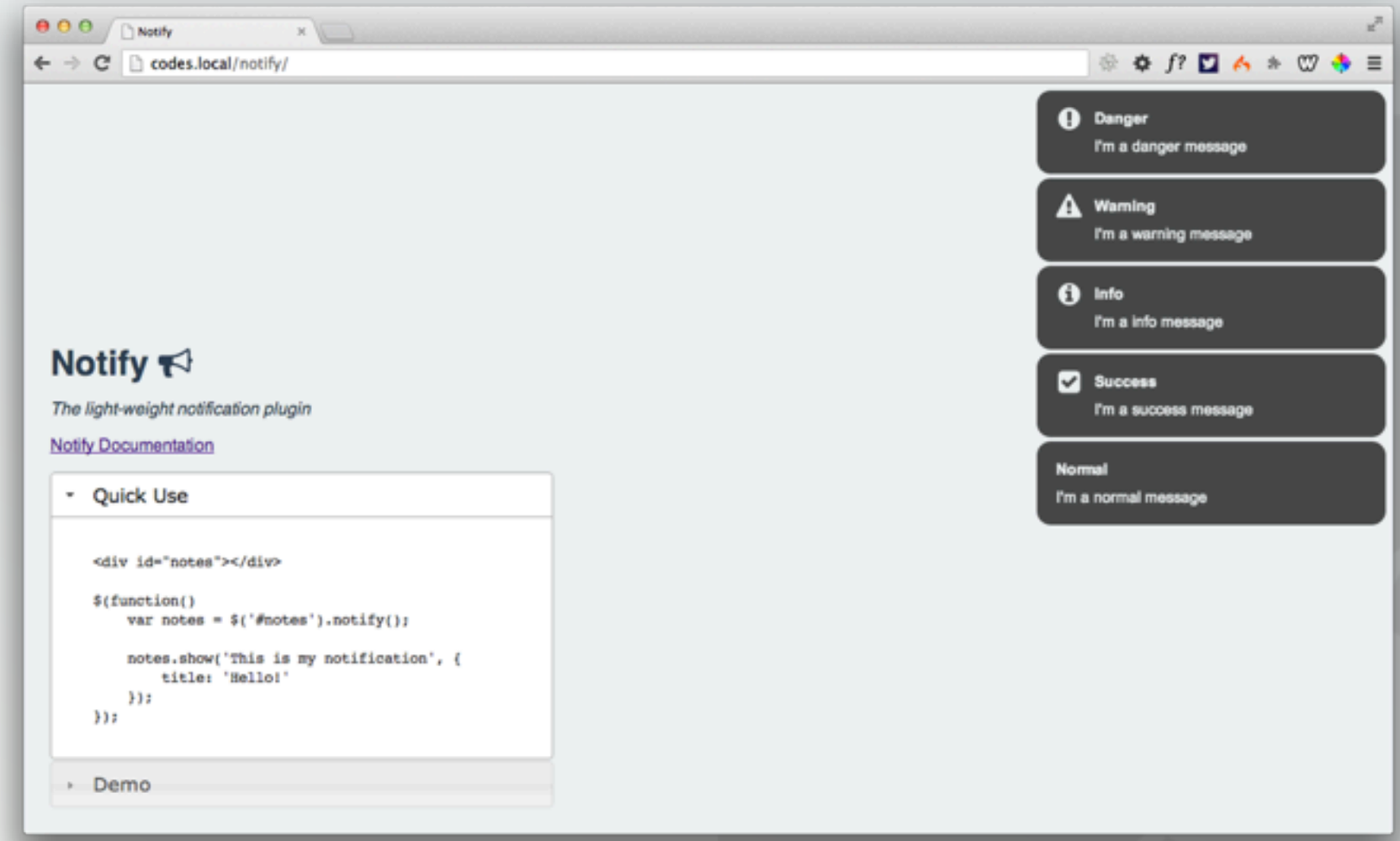
```
$('<span>Adding</span>')
.addClass('adding')
.insertAfter('.wishlist');
$('.wishlist')
.click(function(e) {
doOneUp(this, function() {
$(this).prev().text('Added');
});
e.preventDefault();
})
```

Passing Callbacks
Notice the callback variable that's being passed around in the example? We supply a function as a parameter to our doOneUp code, but we don't do anything with it ourselves; we just pass it along as the callback to jQuery's hide action.
When hide completes, it will run whatever code we gave it. In this case, it's the code to change the link text from Add to wishlist to Added.

```
function doOneUp(which, callback) {
$(which)
.next()
.show()
.animate({
top:"-=50px",
opacity:"toggle"
},
1000,
function() {
$(this)
.css({top: ""})
.hide('slow', callback)
.remove();
});
}
```

dialogs.Notifications

1-up Notifications



week3.**Assignment8**

assignment.0308

Today's Assignment: Forms, Controls & Dialogs **Due:** By End of Lab

- **Make 1 form (must validate), 1 control and 1 Dialog (Growl or 1-Notification)**
- **ALL** html/css/js completed, only 1 stylesheet file for the entire project
- Turn into GitHub Repo: **0308_forms_lastname_firstname_.zip**

This Week's Milestone: Ajax **Due:** Tuesday (By the End of Lab)

- **ALL** html/css markup completed, ajax a must in deliverable
- filler content *(**NO** lorem ipsum)* must be used inside html to test your design
- **ALL** components of your app as HTML (i.e. landing.html, registration.html)
- only 1 stylesheet file for the entire project
- *each html page should **look** like it would when live.*
- Turn into GitHub Repo: **milestone3_ajax_lastname_firstname.zip**