

# HTML Forms

## Web Architecture and Information Management [./] Spring 2009 — INFO 190-02 (CCN 42509)

Erik Wilde, UC Berkeley School of Information

2009-02-11



<http://creativecommons.org/licenses/by/3.0/>

This work is licensed under a [CC Attribution 3.0 Unported License](http://creativecommons.org/licenses/by/3.0/) [http://creativecommons.org/licenses/by/3.0/]

## Contents

• Abstract	2
• 1 Forms Basics	
◦ HTTP Web Services	4
◦ Forms Mechanics	5
◦ Forms Markup	6
◦ Forms Elements (User View)	7
◦ Forms Elements (Source View)	8
◦ Date Entry	9
◦ Forms and GET	10
◦ Forms and POST	11
◦ POST Form Processing	12
◦ Processing of Form Data	13
• 2 Structuring Forms	
◦ Form Usability	15
◦ Labels	16
◦ Fieldsets	17
◦ Tabbing in Forms	18
◦ Disabled and Readonly Controls	19
◦ Disabled and Readonly Controls Display	20
◦ Markup for Disabled/Readonly Controls	21
• Conclusions	22

## Abstract (2)

---

This lecture introduces *HTML Forms*, a way how an HTML page can provide input fields, so that users can provide data to a Web-based application. HTML forms are regular HTML pages (i.e., using regular HTML structures), but they also contain special HTML elements for data entry. Most importantly, each form contains instructions on how to submit the entered data, and the browser will use that information to compose a request containing all the data of the form submission.

# Forms Basics

---

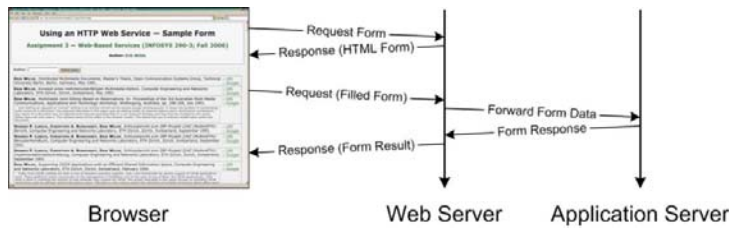
## HTTP Web Services (4)

---

- Services can be provided through URI/HTTP
  - URI-based services need input as a [query string](#) [Web Foundations (URI and HTTP); Query Information (1)]
    - the question is how the user gets information into the URI
- HTML forms provide an interface for assembling query strings
  - users fill out a form providing several fields
  - the browser submits the entered information by HTTP to a URI
  - the result of the request is displayed to the user
- HTTP has two different methods for submitting data
  - [GET](#) [Forms and GET (1)] encodes the data as a URI query string
  - [POST](#) [Forms and POST (1)] encodes the data as HTTP request entity

## Forms Mechanics (5)

- HTML forms are normal Web pages (using form elements)
- The process receiving the form data produces a result page



## Forms Markup (6)

- All form elements must be inside a form element
  - specifies the URI for submitting the form values (`action="URI"`)
  - specifies the method for submitting the form values (`method="GET [Forms and GET (1)]|POST [Forms and POST (1)]"`)
- *form* contains regular HTML markup and form elements
  - the regular HTML markup creates the form's layout (table, list, texts)
  - the form elements create the controls for acquiring input data
- Each form should have a *submit button*
  - when pressing this button, the form values are sent to the action URI
  - without such a button, the form values cannot be submitted

## Forms Elements (User View) (7)

- HTML provides a small set of form controls
- Sufficient for many applications

Text:

Password:

Checkbox:

Radio Button:

Text Areas:

Selection:

Multiple Selection:

File Upload:

Hidden:

Submit:

## Forms Elements (Source View) (8)

```
<form action="http://stevex.net/dump.php" method="POST"
enctype="multipart/form-data"><table>
  <tr><td>Text:</td><td><input type="text" name="text" value="text
input"/></td></tr>
  <tr><td>Password:</td><td><input type="password" name="password"
value="hidden text"/></td></tr>
  <tr><td>Checkbox:</td><td><input type="checkbox" name="check"
value="1"/> <input type="checkbox" name="check" value="2"/> <input
type="checkbox" name="check" value="3"/></td></tr>
  <tr><td>Radio Button:</td><td><input type="radio" name="radio"
value="1"/> <input type="radio" name="radio" value="2"/> <input type="radio"
name="radio" value="3"/></td></tr>
  <tr><td>Text Areas:</td><td><textarea name="textarea" rows="2"
cols="20"/></td></tr>
  <tr><td>Selection:</td><td><select name="select"><option
selected="selected">XML</option><option>SGML</option></select></td></tr>
  <tr><td>Multiple Selection:</td><td><select name="mselect"
multiple="multiple"><option>242</option><option>290-3</option>
<option>290-13</option></select>
  <tr><td>File Upload:</td><td><input name="file" type="file"/>
</td></tr>
  <tr><td valign="top" align="right">Hidden:</td><td><input
type="hidden" name="hidden" value="hidden input"/></td></tr>
  <tr><td>Submit:</td><td><input name="submit" type="submit"/></td></tr>
</table></form>
```

## Date Entry (9)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>JavaScript Datepicker</title>
<script type="text/javascript" src="date-picker.js"></script>
<link rel="stylesheet" type="text/css" href="date-picker.css">
</head>
<body>
<form action="http://stevex.net/dump.php" method="GET">
<input type="text" name="date">
<input type="button" value="calendar" onclick="displayDatePicker('date');">
<input type="submit" value="submit date">
</form>
</body>
</html>
```

## Forms and GET (10)

- Limited to string-oriented form values
  - but HTML forms also allow file upload (this requires POST)
- All values of all form input fields are collected
  - for text and selection fields, this is one input field
  - for checkboxes and radio buttons, this collects the selected fields
- The browser composes a URI query string
  - the form submission is a set of name/value pairs (names may appear more than once!)
  - using URI query string notation, it is appended to the URI of the form's action
- GET is good!
  - URI-encoded queries can be bookmarked and otherwise reused (e.g., cached)
  - if possible, use GET when implementing a form

## Forms and POST (11)

- GET encodes the values in the URI
  - for file uploads, this is not possible
  - HTTP's POST request method can upload data
- POST sends a request containing an entity
  - the HTTP request then looks similar to a response (header fields and entity)
  - the receiving process (the Web server) accepts the POST body
- Entities can use any format (it is specified in a header field)
  - just like e-mails, entities can have multiple parts
  - the parts are separated using the standard MIME mechanism

## POST Form Processing (12)

- POST is used if the form specifies it
  - it can (but should not) be used for non-file forms
  - it should be used for file upload forms (otherwise, only the name is uploaded)
- File upload forms must specify the appropriate encoding
  - "application/x-www-form-urlencoded" is the default (values in the entity)
  - "multipart/form-data" is required for file upload (multipart form data)
- The server side must be prepared to receive POST requests
  - it must parse the entity rather than the URI's query string
  - form values can then be extracted from the entity
  - some environments (e.g., PHP) allow to handle GET/POST transparently

## Processing of Form Data

**(13)**

- Form data is always encoded
  - as a query string when using GET
  - in an encoded entity when using POST
  - in a multipart entity when using POST with multipart/form-data
- Parsing the form data should be done by existing software
  - most Web-aware programming environments provide this functionality
  - PHP allows access through different mechanisms

```
<p>Query Parameter Test:</p>
<p>test: <?php echo $_REQUEST["test"]; ?></p>
<p>name: <?php echo $_REQUEST["name"]; ?></p>
<p>field: <?php echo $_REQUEST["field"]; ?></p>
```

# Structuring Forms

## Form Usability

**(15)**

- HTML forms are very loosely structured
  - form somewhere representing the container
  - inside the form a random collection of HTML and form inputs
- Visually, the structure often is (and should be) easy to see
  - for non-visual access, more structure must be provided
  - accessibility has become a major issue on the Web
- Accessibility has many different facets
  - voice browsers must be able to read aloud Web forms
  - gateways should be able to intelligently re-structure Web forms

## Labels (16)

- Label and form control are not connected by HTML

```
<tr><td>Text:</td><td><input type="text" name="text"/></td></tr>
  <tr><td>Password:</td><td><input type="password" name="password"/>
</td></tr>
```

- The label element allows to make this connection
  - it connects a form control with the describing label
  - this association is now accessible to clients for processing

```
<tr>
  <td><label for="textctrl">Text:</label></td>
  <td><input type="text" name="text" id="textctrl"/></td>
</tr>
<tr>
  <td><label for="pwdctrl">Password:</label></td>
  <td><input type="password" name="password" id="pwdctrl"/></td>
</tr>
```

## Fieldsets (17)

- Complex forms may need structuring
  - groups of controls for subsets of the collected data
  - this structure should be represented in markup

```
<fieldset><legend>Billing</legend>billing form controls ...</fieldset>
  <fieldset><legend>Shipping</legend> shipping form controls ...
</fieldset>
```

- Excellent example for HTML's markup design philosophy
  - if a client does not support fieldsets, the elements are ignored
  - the title of the fieldset will be displayed, because it is text

```
Billing
  billing form controls ...
```

```
Shipping
  shipping form controls ...
```

## Tabbing in Forms (18)

- Tabbing is a very convenient way of navigating a form
  - after completing one field, users should be taken to the next
  - the order should be defined by the form creator, not by accident
- the `tabindex` attribute defines the tabbing order
  - it contains a number which is interpreted relative to other numbers
  - all form controls may carry a `tabindex` attribute
- `tabindex 1-9`:

## Disabled and Readonly Controls (19)

- In complex scenarios, certain controls may be disabled or readonly
  - based on a workflow, some controls may not apply in all cases
  - for the sake of a consistent view, they should still be included in the interface
- Disabled controls are not used
  - they cannot be tabbed to and never receive focus
  - their value is not included in the form's submission data
- Readonly controls cannot be changed
  - they can be tabbed to and may receive focus
  - their *value* may not be changed (important for radio buttons and checkboxes!)
  - their value is included in the form's submission data
- Important: Never trust the Browser!

## Disabled and Readonly Controls Display (20)

	Normal Control	Disabled Control	Readonly Control
Text:	<input type="text" value="text input"/>	<input disabled="disabled" type="text" value="text input"/>	<input readonly="readonly" type="text" value="text input"/>
Password:	<input type="password" value="password"/>	<input disabled="disabled" type="password" value="password"/>	<input readonly="readonly" type="password" value="password"/>
Checkbox:	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/> <input checked="" type="checkbox"/> <input disabled="disabled" type="checkbox"/>	<input type="checkbox"/> <input checked="" type="checkbox"/> <input disabled="disabled" type="checkbox"/>
Radio Button:	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	<input type="radio"/> <input checked="" type="radio"/> <input disabled="disabled" type="radio"/>	<input type="radio"/> <input checked="" type="radio"/> <input disabled="disabled" type="radio"/>
File Upload:	<input type="text" value=""/> <input type="button" value="Browse..."/>	<input type="text" value=""/> <input disabled="disabled" type="button" value="Browse..."/>	<input type="text" value=""/> <input disabled="disabled" type="button" value="Browse..."/>
Text Areas:	<input type="text" value="initial text"/>	<input disabled="disabled" type="text" value="initial text"/>	<input readonly="readonly" type="text" value="initial text"/>
Selection:	<input type="text" value="XML"/>	<input disabled="disabled" type="text" value="XML"/>	
Multiple Selection:	<input type="text" value="242"/> <input type="text" value="290-3"/> <input type="text" value="290-13"/>	<input type="text" value="242"/> <input type="text" value="290-3"/> <input disabled="disabled" type="text" value="290-13"/>	[ not supported ]

## Markup for Disabled/Readonly Controls (21)

```
<table style="margin : 2% ; width : 90% ; " rules="groups" cellpadding="5">
  <thead><tr>
    <td></td>
    <th>Normal Control</th>
    <th>Disabled Control</th>
    <th>Readonly Control</th>
  </tr></thead>
  <tbody>
    <tr><td valign="top" align="right">Text:</td><td><input type="text"
      name="text1" value="text input"></td><td><input disabled="disabled"
      type="text" name="text2" value="text input"></td><td><input
      readonly="readonly" type="text" name="text3" value="text input"></td></tr>
    <tr><td valign="top" align="right">Password:</td><td><input
      type="password" name="password1" value="hidden text"></td><td><input
      disabled="disabled" type="password" name="password2" value="hidden text">
      </td><td><input readonly="readonly" type="password" name="password3"
      value="hidden text"></td></tr>
    <tr><td valign="top" align="right">Checkbox:</td><td><input
      type="checkbox" name="check1" value="1"> <input type="checkbox" name="check1"
      checked="checked" value="2"> <input type="checkbox" name="check1" value="3">
      </td><td><input disabled="disabled" type="checkbox" name="check2" value="1">
      <input disabled="disabled" type="checkbox" name="check2" checked="checked"
      value="2"> <input disabled="disabled" type="checkbox" name="check2" value="3">
      </td><td><input readonly="readonly" type="checkbox" name="check3" value="1">
      <input readonly="readonly" type="checkbox" name="check3" checked="checked"
      value="2"> <input readonly="readonly" type="checkbox" name="check3"
      value="3"> !</td></tr>
  </tbody>
</table>
```

## Conclusions

**(22)**

- HTML forms allow data entry on Web pages
- Only a small number of form controls are available
- Form submission is just another HTTP request
- Forms should be structured to be more accessible