

USABILITY BEST PRACTICES

Usability does not stop with the user experience. Extending usability to our code can provide a number of benefits – higher code readability, quicker changes, and simplifying the process of ensuring user usability.

XHTML

The XHTML 1.0 standard has been released by W3C. Any markup code should be written to this standard. This will strengthen the structure of pages, improve code readability, and help ensure forward-compatibility. Some important points in the XHTML specification can be found below, while the full specification can be found at <http://www.w3.org/TR/xhtml1/>.

- A `DOCTYPE` declaration must be included before the `HTML` element. This will generally be in the template. [\[DOCTYPE Example\]](#)
- Every element must be terminated or closed. For non-empty elements, this means using a closing tag at the end of the element. Empty elements – such as `br`, `hr`, and `img` – must use a trailing `/` before the end of the tag to prevent uncertain results in some browsers. A space must precede the trailing `/`. [\[XHTML Example\]](#)
- Elements must nest properly. Such that a child element is closed before its parent element is closed. [\[XHTML Example\]](#)
- Elements should be indented according to their nesting. A child element should be indented one tab more than its parent element on a preceding line. [\[XHTML Example\]](#)
- Element and attribute names must be in lower case. Enumerated attribute values should also be in lower case. This is necessary to allow XML parsing of the document and helps improve code readability. *Exception: Proper capitalization of .Net control element and attribute names is acceptable and encouraged since they will not be rendered on the client side. Proper capitalization of these elements will help to differentiate them from static elements.* [\[XHTML Example\]](#)
- All attributes must have an accompanying value. Minimized attributes, such as `checked` in `input` elements of type `radio` or `checkbox`, should have a value of the attribute name.
- Attribute values must always be quoted, even those that appear to be numeric. This improves the structure of the element for parsing. [\[XHTML Example\]](#)
- Script elements for client-side scripts should have a defined type and language. Additionally, the contents of the script element should be contained within an HTML comment to hide the script from older browsers. [\[Script Example\]](#)

CSS

While W3C is currently working on the level 3 specification for CSS, widespread support only exists for the level 1 specification. Some limited support exists for the level 2 revision 1 specification. More information can be found at <http://www.w3.org/Style/CSS/>. The specifications cover the syntax of selectors, properties, values, and their behaviors, but realize that not all aspects of the specifications are supported by even the most modern browsers. Below are some best practices to use when writing CSS improve code readability and decrease the time necessary for changes.

- All CSS code should be contained within linked style sheets unless it depends upon programmatic conditions. This centralizes CSS code, allowing for quicker changes.
- Element IDs and classes should be named according to their syntactical purpose, not their definition.
- The selector(s), the opening {, each property, and the closing } should be on separate lines. When it is necessary to write CSS code in a single line, separate each of these components with a space. Each property should be indented one tab. Each property name should be followed by a space after the colon. [\[CSS Example\]](#)
- Elements and identifiers used solely for presentation should be kept to a minimum. Try not to use an extra identifier or class where a more focused selector would suffice.
- Do not allow shorthand properties to decrease code readability. If you need to look up the order of the values or the effect of the values are not apparent, separate properties should be used. Readability is a higher priority than file size.

SEMANTIC MARKUP

HTML was originally developed as a markup language to indicate the structure of the scientific papers being shared on the Internet. As such, there are many elements to define types of items such as headers, paragraphs, and lists. Unfortunately, these semantic elements are often ignored for their generic counterparts.

While the generic elements can be styled to look like headers, paragraphs of content, or list items; they are then dependent on styles for part of their meaning. Styles may fail to load – possibly because a screen reader is being used, the browser does not support CSS, or transfer errors. Below are some key elements to consider when marking up content.

- `h2` – This element should be used for top-level headers within the content (`h1` is reserved for the site, area, or page title). Top-level headers may include content headers, form names, and list names. It should be styled like the `header` style available in the editor. [\[Simple List Example\]](#) [\[List Example\]](#) [\[Form Example\]](#)
- `h3` – This element should be used for second-level headers within the content. Second-level headers may include content sub headers, form section names, and list item titles when items contain various pieces of information. It should be styled like the `subHeader` style available in the editor. [\[List Example\]](#)
- `p` – This element should be used for paragraphs or other logical groupings of content. [\[List Example\]](#)
- `ul` and `ol` – These elements should be used around a list of a single type of item – such as links, form fields, events, news releases, etc. All list elements should have a class indicating the type of items it contains. [\[Simple List Example\]](#) [\[List Example\]](#) [\[Form Example\]](#)
- `li` – This element should be used for an item within a list. List item elements should not need a class other than to specify alternate item styles. [\[Simple List Example\]](#) [\[List Example\]](#) [\[Form Example\]](#)
- `label` – This element must be used to indicate a label for a form field, using the ID of the form field as the value of the `for` attribute. [\[Form Example\]](#)

CONVENTIONS

When using a listing control – such as a Repeater or DataGrid – no code should be used within the item templates. Instead, controls within the templates should be populated by an `OnItemDataBound` function. [\[Repeater Example\]](#)

EXAMPLES

The following examples are intended to illustrate the practices described in the above sections. The example links in the above sections will take you to the accompanying example.

EX. 1 – DOCTYPE

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

[Back to XHTML](#)

EX. 2 – XHTML

```
<p>
  <br />
  This is a <strong>djembe</strong>. It is a drum from
  <a href="http://en.wikipedia.org/wiki/Western_Africa" title="West Africa">
    <em>West Africa</em>
  </a>.
</p>
```

[Back to XHTML](#)

EX. 3 – SCRIPT

```
<script type="text/javascript" language="javascript">
  <!--
  alert('Hello world!');
  //-->
</script>
```

[Back to XHTML](#)

EX. 4 – CSS

```
.personnel h2 {
  margin: 0.5em 0;
  font-size: 14px;
  font-weight: bold;
}
.personnel ul {
  margin: 0;
  padding: 0;
  list-item-type: none;
}
.personnel ul li {
  padding: 3px 8px;
}
.personnel ul li.alt {
  background-color: #C8C8E2;
}
```

[Back to CSS](#)

EX. 5 – SIMPLE LIST

```
<div class="personnel">
  <h2>Our People</h2>
  <ul>
    <li>Gwen Pearson</li>
    <li class="alt">Milton Waddams</li>
    <li>Annabelle Farrell</li>
    <li class="alt">Joshamee Gibbs</li>
    <li>James Edwards</li>
  </ul>
</div>
```

[Back to Semantic Markup](#)

EX. 6 – LIST

```
<div class="personnel">
  <h2>TCG Personnel</h2>
  <ul>
    <li>
      <h3>Jeff Samples</h3>
      <p>Jeff is our fearless leader.</p>
    </li>
    <li class="alt">
      <h3>Jared Tucker</h3>
      <p>Jared comes up with all kinds of cool stuff.</p>
    </li>
  </ul>
</div>
```

[Back to Semantic Markup](#)

EX. 7 – FORM

```
<form id="frmSignup" class="signup" action="/signup.aspx" method="get">
  <h2>Signup</h2>
  <ul class="signupFields">
    <li>
      <span class="formLabel">
        <label for="txtFname">First Name:</label>
      </span>
      <span class="formInput">
        <input type="text" class="text" id="txtFname" name="txtFname" />
      </span>
    </li>
    <li class="alt">
      <span class="formLabel">
        <label for="txtLname">Last Name:</label>
      </span>
      <span class="formInput">
        <input type="text" class="text" id="txtLname" name="txtFname" />
      </span>
    </li>
    <li>
      <span class="formLabel">
        <label for="txtEmail">E-mail:</label>
      </span>
      <span class="formInput">
        <input type="text" class="text" id="txtEmail" name="txtEmail" />
      </span>
    </li>
    <li>
      <span class="formLabel">
        &nbsp;
      </span>
      <span class="formInput">
        <input type="submit" class="btn" id="btnSubmit" name="btnSubmit" />
        <input type="cancel" class="btn" id="btnCancel" name="btnCancel" />
      </span>
    </li>
  </ul>
</form>
```

[Back to Semantic Markup](#)

EX. 8 – REPEATER

```
private void rptResults_ItemDataBound(Object sender, RepeaterItemEventArgs e) {
    if (e.Item.ItemType == ListItemType.Item ||
        e.Item.ItemType == ListItemType.AlternatingItem) {

        if (e.Item.ItemType == ListItemType.AlternatingItem) {
            ((HtmlGenericControl)(e.Item.FindControl("liResult"))).Attributes.Add(
                "class", "alt");
        }

        ((HyperLink)(e.Item.FindControl("hlResult"))).NavigateUrl =
            DataBinder.Eval(e.Item.DataItem, "URI").ToString();

        ((HyperLink)(e.Item.FindControl("hlResult"))).Text =
            DataBinder.Eval(e.Item.DataItem, "Title").ToString();

        ((Label)(e.Item.FindControl("lblResultDescr"))).Text =
            DataBinder.Eval(e.Item.DataItem, "Summary").ToString();

        ((HyperLink)(e.Item.FindControl("hlResultUrl"))).NavigateUrl =
            DataBinder.Eval(e.Item.DataItem, "URI").ToString();

        ((HyperLink)(e.Item.FindControl("hlResultUrl"))).Text =
            DataBinder.Eval(e.Item.DataItem, "URI").ToString();
    }
}

<div class="searchResults">
    <asp:Repeater id="rptResults" runat="server" OnItemDataBound="rptResults_ItemDataBound">
        <HeaderTemplate>
            <h2>Search Results</h2>
            <ul class="search">
        </HeaderTemplate>
        <ItemTemplate>
            <li id="liResult" runat="server">
                <h3><asp:HyperLink ID="hlResult" runat="server" /></h3>
                <p><asp:Label ID="lblResultDescr" runat="server" /></p>
                <p class="contentSmall">
                    <asp:HyperLink ID="hlResultUrl" runat="server" />
                </p>
            </li>
        </ItemTemplate>
        <FooterTemplate>
            </ul>
        </FooterTemplate>
    </asp:Repeater>
</div>
```

[Back to Conventions](#)