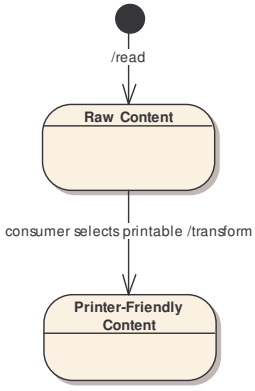
Printer-Friendly Article

Remove stylized formatting and preserves the pure article text and images, allowing the text and graphics fit the width of any printer.



Background

A consumer is reading a page from a Web site. The page was created using *Dynamic Web Page (page #)* and *Stylized Page (page #)*. The think that the information on the page is helpful and they would like to make a hardcopy of it. However, the page design and layout favor online readability and aesthetics, not printability.

The business needs to provide the means to reformat the page to make it compatible with the wide variety of printers that consumers may use.

Value and Benefits

We could simply design two pages, one for online viewing and one for print. However, that is a time consuming, labor intensive solution. Making the second article on our site doesn't really reuse and effort expended to make the first article printable. The same goes for the third, fourth, fifty, and well, you get the point.

There is value in reusing the same automated engine that produced the *Stylized Page (page #)* to create the bland page too. The benefit is reuse of tools and expertise, creating a solution that works time and again, and providing the consumer a way use the information in a way they are comfortable with.

Putting It to Work

Place one or more links on every page for which you will provide a printer-friendly version. The link can be as simple as this:

Printer-Friendly Version

This is a minor point, but many times I find this link only at the top of the page. It is many times not until I have reached the bottom of the page that I decide to print it. So why not place two such links on the page, one at the top and one at the bottom?

Create a transformation description that takes the raw content, both text and graphics, as input and produced *bland* content as output. What is meant by bland content is

formatting the page so that text lines break according to the browser window width, not according to a ridged design layout. When lines break naturally per the browsers width, they will also naturally fit the printed page's width.

Using HTML markup, this means placing each paragraph between separate paragraph tags (`<p>…</p>`). To make some text, such as the title and headings, standout, use the HTML heading markup (`<h1>…</h1>`, etc.) or appropriately sized graphic images. There is no reason to use HTML tables to format the text. In fact any such constructs could interfere with allowing the browser to format the page for the printer.

If your consumers are printing a standard US Letter size page, or to the more international A4 page, make sure that graphic illustrations are not more than around 6 inches or XX cm wide. Also limited the height of illustrations to those that are clear, but that are no higher than, say, half or three-quarters of the page height (at most!). For both width and height considerations, remember that the page margins will reduce the real estate you actually have to print on. All graphics should be on the left margin (for left-to-right languages, anyway), or centered (`align="middle"`).

Banner and sidebar advertisements are probably inappropriate for printer-friendly pages. Consumers will probably not find a good way to respond to ads since they are not reading the page online. They will also find it more challenging to filter ads from article illustrations as they read the printed page.

Make sure that your raw content in some way indicates that it can be transformed into a *Printer-Friendly Article*. This may be indicated with a special indicator, or by some value that is native to the printable articles. If you use XML and XSL to generate pages, you might include a `<printable/>` element in your DTD or schema. If you are dealing with legacy content that you cannot easily add logic to, you might probe for one or more native nodes and/or content that all printable articles would have at least one of. This will allow *Stylized Page (page #)* to generate **Printer-Friendly Version** links automatically.

When the consumer selects the **Printer-Friendly Version** link, run the raw content through your transformation engine using the "bland" transformation description. When the page response is generated return it to the browser.

Example

There are a few suitable implementation strategies. I present one here in detail. It uses a simple Java utility class that creates HTML from text content. It is simple so as not to cloud the point of the pattern:

```
public class TextToHtmlTransformer implements TransformerIF
{
  public TextToHtmlTransformer()
  {
      super();
  }

  public byte[] transform(TransformerInputSourceIF anInputSource)
    throws TransformationException
  {
    StringBuffer tempHtmlDoc = new StringBuffer();
    String tempTitle = (String) anInputSource.getAttribute("title");

    tempHtmlDoc.append("<html>\n");
    tempHtmlDoc.append("<head>\n");
    tempHtmlDoc.append(
```

```
      "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\">\n");
    tempHtmlDoc.append("<title>");
    tempHtmlDoc.append(tempTitle);
    tempHtmlDoc.append("</title>\n");
    tempHtmlDoc.append("</head>\n");
    tempHtmlDoc.append("<body bgcolor=\"#FFFFFF\">\n\n");
    tempHtmlDoc.append("<h1>" + tempTitle + "</h1>");

    String tempTextDoc = new String(anInputSource.getContent());

    StringTokenizer tempStrTok = new StringTokenizer(tempTextDoc, "\r\n");

    while (tempStrTok.hasMoreTokens())
    {
      String tempPara = tempStrTok.nextToken();

      tempHtmlDoc.append("<p>");
      tempHtmlDoc.append(tempPara);
      tempHtmlDoc.append("</p>");
      tempHtmlDoc.append("\n");
    }

    tempHtmlDoc.append("\n");
    tempHtmlDoc.append("</body>\n");
    tempHtmlDoc.append("</html>\n");

    return tempHtmlDoc.toString().getBytes();
  }
}
```

This `TextToHtmlTransformer` does not handle graphic images, it only transform plain ANSI text with line breaks (such as a source code listing or README file). Significant is that the HTML output is generated in a stoically bland way. The title text is marked up with an HTML H1 style (heading level 1), and each text paragraph is set in its own HTML paragraph.

Key to this implementation is two interfaces, `TransformerIF` and `TransformerInputSourceIF`. `TransformerIF` defines the contract between transformer clients and transformers. It defines a single method that must be implemented by all transformers:

```
public interface TransformerIF
{
  public byte[] transform(TransformerInputSourceIF anInputSource)
        throws TransformationException;
}
```

The `transform()` method answers an array of `bytes` that contains the results of the transformation. It takes a `TransformerInputSourceIF` instance as a parameter:

```
public interface TransformerInputSourceIF
{
  public Object getAttribute(String aName) throws TransformationException;

  public Object getAttribute(String aName, int anIndex) throws TransformationException;

  public byte[] getContent() throws TransformationException;
}
```

This interface allows the transformer to read the content and metadata about the content. For example, class `TextToHtmlTransformer` uses the `getContent()` method to read the content as bytes. The overloaded `getAttribute()` methods

provide metadata about the content. Class `TextToHtmlTransformer` uses the non-indexed form of the method to get the content's title. The indexed form of `getAttribute()` is used to get elements of metadata arrays. The implementers must supply the data.

Another implementation strategy that works well uses XML and XSL transformations. This strategy is presented in *Stylized Page (page #)*. It is probably the most versatile method and supports a rich input document model. It may be a bit slower, however. In fact the input document stream would not change from that presented in *Stylized Page*. You would basically just dumb-down the style sheet used to transform the XML document.

Consequences

These are the competing forces within the *Printer-Friendly Article* solution pattern.

- ***Choose the Right Pages***: It is not a good idea to provide printer-friendly links on every page on your site. Make sure you have some means of identifying all such articles, and excluding pages that should not be specially formatted for print.

Frameworks and Tools

- **Frameworks and Tools**: The same tools used by *Stylized Page (page #)* are appropriate for use by this pattern.