HTML Parser

HTML Parser is a Java library used to parse HTML in either a linear or nested fashion. Primarily used for transformation or extraction, it features filters, visitors, custom tags and easy to use JavaBeans. It is a fast, robust and well tested package.

Welcome to the homepage of HTMLParser - a super-fast real-time parser for real-world HTML. What has attracted most developers to HTMLParser has been its simplicity in design, speed and ability to handle streaming real-world html.

The two fundamental use-cases that are handled by the parser are extraction and transformation (the syntheses use-case, where HTML pages are created from scratch, is better handled by other tools closer to the source of data). While prior versions concentrated on data extraction from web pages, Version 1.4 of the HTMLParser has substantial improvements in the area of transforming web pages, with simplified tag creation and editing, and verbatim toHtml() method output.

In general, to use the HTMLParser you will need to be able to write code in the Java programming language. Although some example programs are provided that may be useful as they stand, it's more than likely you will need (or want) to create your own programs or modify the ones provided to match your intended application.

To use the library, you will need to add either the htmllexer.jar or htmlparser.jar to your classpath when compiling and running. The htmllexer.jar provides low level access to generic string, remark and tag nodes on the page in a linear, flat, sequential manner. The htmlparser.jar, which includes the classes found in htmllexer.jar, provides access to a page as a sequence of nested differentiated tags containing string, remark and other tag nodes. So where the output from calls to the lexer nextNode() method might be:

```
<html>
<head>
<title>

"Welcome"
</title>
</head>
<body>
etc...
```

The output from the parser NodeIterator would nest the tags as children of the <html>, <head> and other nodes (here represented by indentation):

```
<html>
    <head>
        <title>
            "Welcome"
            </title>
        </head>
    <body>

        etc...
```

The parser attempts to balance opening tags with ending tags to present the structure of the page, while the lexer simply spits out nodes. If your application requires only modest structural knowledge of the page, and is primarily concerned with individual, isolated nodes, you should consider using the lightweight lexer. But if your application requires knowledge of the nested structure of the page, for example processing tables, you will probably want to use the full parser.

Extraction

Extraction encompasses all the information retrieval programs that are not meant to preserve the source page. This covers uses like:

- text extraction, for use as input for text search engine databases for example
- link extraction, for crawling through web pages or harvesting email addresses
- screen scraping, for programmatic data input from web pages
- resource extraction, collecting images or sound
- a browser front end, the preliminary stage of page display
- link checking, ensuring links are valid
- site monitoring, checking for page differences beyond simplistic diffs

There are several facilities in the HTMLParser codebase to help with extraction, including filters, visitors and JavaBeans.

Transformation

Transformation includes all processing where the input *and* the output are HTML pages. Some examples are:

- URL rewriting, modifying some or all links on a page
- site capture, moving content from the web to local disk
- censorship, removing offending words and phrases from pages
- HTML cleanup, correcting erroneous pages
- ad removal, excising URLs referencing advertising
- conversion to XML, moving existing web pages to XML

During or after reading in a page, operations on the nodes can accomplish many transformation tasks "in place", which can then be output with the toHtml() method. Depending on the purpose of your application, you will probably want to look into node decorators, visitors, or custom tags in conjunction with the PrototypicalNodeFactory.