Wsmake User Manual

Michael L. Brownlow[1]

November 24, 2002

[1]http://www.wsmake.org/~mike/

Contents

List of Figures

Chapter 1

Introduction

This text is the user manual for Wsmake, a website preprocessor. The goal for the reader of this text should be to understand what Wsmake is used for and how to use it. This text can also be used as a reference starting in Chapter 4.

The reader should have a basic understanding of computers before reading this manual. In particular, you should be comfortable with a shell environment[1] and a text editor[2].

If you are looking for instructions on installing and maintaining the Wsmake program, please see the installation manual.

The rest of this introduction contains a small history of Wsmake and plans for the future. Skip to Chapter 2 if you are in a hurry to get started with Wsmake.

1.1 History

In the years of the web gold rush in the last millenium I was creating websites for various companies, groups and individuals. Like many others, I noticed that for many websites it was convenient to have a common interface on each page to link them all together. Usually this involved three to six top level categories. This tree of categories increased in complexity as subcategories of the top level categories were added. At first it was easy to duplicate the common interface to each page, but not for long. So wsmake was born early in 1999 as a small shell script to address this problem. The script worked quite well for many hours, but eventually more complex things were desired and wsmake became a Perl script named "webthemes.pl." Yet it lacked still. About this time I was taking a C++ programming course at the University of Texas at Arlington and was inspired to rewrite Wsmake from scratch using some parts in C and some in C++. The new version was named "webmake." While registering Wsmake with software news

[1] Like bash or tcsh.

[2] Like emacs or vim.

websites like freshmeat[3] I found there was already a program named "webmake" [4] out there, so I renamed my webmake to the now very elegant and obscure name "wsmake." Now Wsmake has its own domain, wsmake.org, complete with CVS, snapshots, packaged binaries, and more.

1.2 The Future

Over time Wsmake's purpose has widened. In the beginning it targeted replication of common data from a single source. This feature has remained intact and represents the foundation for where Wsmake is today. I.e., it is data centric. Now Wsmake has goals in the areas of generalized data manipulation and programmability through abstraction and hierarchical scoping. Wsmake has experienced steady growth since the year 2000. Both big corporations and individuals alike have found it to be a useful and time-saving tool. Current plans for the future include a wide variety of things from new user interfaces to better scripting support. This manual will be maintained to cover all of these new features as they are implemented.

[3] http://freshmeat.net/
[4] http://webmake.taint.org/

Chapter 2

Basics

In this chapter I will introduce the basics of how Wsmake works. The goals are understanding data flow, configuration, and usage. First, however, I will introduce some terminology that should make the rest of the text easier to read and understand.

2.1 Terminology

In general, Wsmake takes a set of data from an input location, processes it, and then places it in an output location. It does this based on a configuration and on the data itself.

The configuration specifies where the input data is, where the output data should go, and how to process it. In addition, the data itself may specify how to process parts of itself.

These various functionalities can be grouped as follows:

- Data Description Tree
- Extensions
- Substitution

These functions used together act as a *filter* for your data.

2.1.1 Data Description Tree

The Data Description Tree (or DDT for short), is an hierarchy of data scopes with statements. This tree is used to determine where the input data is, where it should go, and how to arrange data for output. It does this based on the order and depth of the data scoping.

The DDT is described in more detail in section 2.3.2.

2.1.2 Extensions

With the appropriate configuration, external programs of your choice may be used to modify the data as it is being processed.

Extensions are described in detail in section 2.3.3.

2.1.3 Substitution

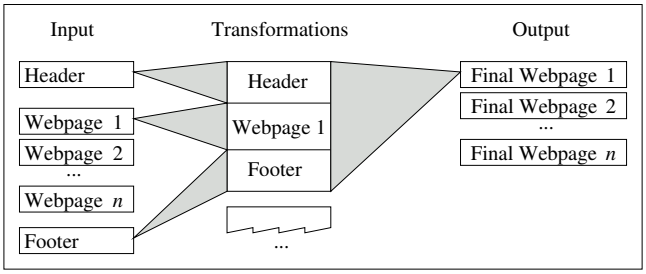
One of the primary benefits of a pre-processor like Wsmake is the ability to do keyword substitution. Wsmake utilizes a regular expression approach to implement this. Each keyword is a regular expression and it is given an associated substution value. These substitutions can be arranged throughout the DDT.

Substitutions are described in detail in section 2.3.4.

2.2 Data Flow

I'm going to introduce data flow with an example. Let's say that you are maintaining a website. Maybe it is a personal website, or a company's intranet website. In any case, you will probably recognize the need for similar features on each webpage. In particular, the header and footer data for each webpage may be very similar in format. However, you do not want to be burdened with with manual replication of that header and footer information to each webpage, especially when the information changes. You would rather maintain that in a single place. To do this implies the need for a transformation sequence of the data, or *data flow*. For this example, we could say that there are certain data items that represent each web page and the header and footer. These data items start the data flow and are recognized as *input* to the flow. The transformations begin after this starting state by modifying each web page in sequence by prepending the header and appending the footer. To complete the flow, each page needs to be placed into an output location. These transformed pages represent the *output* from the flow. Figure 2.1 shows the data flow of this example.

Figure 2.1: Website Example Data Flow



This transformation shows, in its very basic form, what Wsmake can do. The statements mentioned in Section 2.1.1 are the tools to implement the tranformation abstraction. Now I will describe in detail the input and output.

2.2.1 Input

All input to Wsmake begins with one or more configuration locations. The configuration can be specified as standard input or any other file whose data conforms to the configuration specification for Wsmake.

The default behavior of Wsmake is to first read as input the files specified on the command line (See Section 2.4.1). If this is not done, Wsmake will look in the current working directory for a file named `wsmakefile`. If that file is not found, it looks for all files that have a suffix of `.ws`, also in the current working directory. As a last resort, when none of the above types of files are found, it begins to read from standard input.

When the data is ready for processing it is scanned for tokens by a lexer[1]. At this point it is wise to point out that tokens are mostly typical with Wsmake. By this I mean that there are some tokens that are not typical, mainly nested scripts. These tokens are actually replaced by their output and then read by the lexer. I'll describe that in the next section. Another example of this type of token would be an *include* directive, which causes another file's contents to be placed at that point in the input to be read. During all of this the lexer provides the tokens for a parser to match against a set of rules in *Backus-Naur Form*.[2] Ultimately this means the input must conform to the Wsmake grammar, described in Sections 2.3 and 4.1.

Now that the data has been read and it matches the grammar, processing begins and the data items I mentioned in Section 2.2 will be read. These items cause a special type of input to be processed. Different from the configuration input, the data items are the content that you want to transform. It is possible to further embed configuration in the data items, and they will also be read as the primary configuration was. Any configuration in data items is replaced by its output, which I will describe next.

2.2.2 Output

Output from the data flow of Wsmake is placed in certain places which are specified in the configuration.

2.3 Configuration

The configuration tells Wsmake how to process a DDT. Multiple DDTs can be described in a single configuration.

[1] GNU Flex to be precise
[2] Yep, with GNU Bison

2.3.1 Data Language

2.3.2 Data Description Tree

2.3.3 Extensions

Two basic modes are facilitated by Wsmake for this purpose: standard input, and temporary files. With standard input, the script is fed to the external program's *stdin*. Using temporary files, the script is placed in a temporary file and no data is fed to the external program's *stdin* . In both cases, the location of the data sources can be provided on the command line to the program. The data sources include: the script (if not doing standard input), configuration file, and any other related data.

Standard Input

Temporary Files

2.3.4 Substitution

2.4 Usage

2.4.1 Command Line

2.4.2 Text Interface

2.4.3 Graphical Interface

Chapter 3

Advanced Features

3.1 References

3.2 Dependencies

3.3 Cloning

Chapter 4

Reference

4.1 Configuration Syntax

4.1.1 Keywords

4.1.2 Grammar

4.1.3 Blocks

4.1.4 Statements

4.1.5 Commands

4.1.6 Scripts

Index