Biloba STX
Expert's Guide

Version 1.0
17-Apr-2004

Contents

List of Figures

Chapter 1

Introduction

1.1 About This Document

This is the *Expert's Guide* to the Biloba document formatting system. It explains the setup process, program internals and how to write extension modules for the Biloba document formatting system. Formatting commands the average user does not require most of the time are also covered. For a general introduction to document formatting using Biloba, please refer to the *User's Guide*.

Biloba is a non-interactive document formatting system specifically designed for documents on the Web. It was developed as a project completed as part of the requirements for the BSc. (Hons) Computer Studies by Viktor C. Pavlu under the supervision of Carlton McDonald at the University of Derby in the years 2003-2004.

Chapter 2

Setup

2.1 Web Server Installation

This section assists you with setting up Biloba in combination with the Apache HTTP Server. It assumes no prior knowledge of the Apache HTTP Server, however the information on Apache's '`httpd.conf`' configuration file presented here is limited to the minimum required for Biloba. This guide is not intended as replacement for the extensive documentation available on the Apache HTTP Server which is highly recommended before deploying the Web server in a production environment. The documentation is available at `http://httpd.apache.org/docs/`. Note that it is discouraged to use the Biloba prototype in a production environment, as it can not be guaranteed to be safe. For example, Biloba does not check that the requested documents are located within the DocumentRoot — all files readable to Biloba can be accessed through the HTTP server by clever use of the query string. Making the program source available via the HTTP server is also considered to be harmful due to security concerns! Later releases will fix this.

First you have to download and install the Apache HTTP Server which is available at `http://httpd.apache.org/download.cgi`. You will also require an interpreter for the REBOL programming language which is available at `http://www.rebol.com/platforms.shtml`. Installation files for a Microsoft Windows environment can also be found on the accompanying CD in the directory '`w32setup/`'.

Install the HTTP server using the setup program. Then add the following entries to your '`httpd.conf`' located in '`conf/`' inside your Apache installation directory. Search for the '`AddHandler`' directive and add the extension '`.r`' for REBOL files. This will enable '`.r`' files as CGI scripts (see Fig. 2.1).

Then add the following lines to your configuration (see Fig. 2.2) to enable the execution of CGI scripts in the directory '`biloba/`' within your document root. The document root is the directory from which the server obtains the documents

Figure 2.1: Enable .r Files as CGI Scripts

```
AddHandler cgi-script .cgi .r
```

Figure 2.2: Directory Entry That Enables the Execution of CGI Scripts

```
<Directory "YOURDOCROOTHERE/biloba">
  Options ExecCGI
  AllowOverride None

  Order allow,deny
  Allow from all
</Directory>
```

the clients request. By default it is located in the directory '`htdocs/`' within your Apache installation, but can be changed to any directory you like. Replace the text '`YOURDOCROOTHERE`' with your actual path to your document root which is specified by the configuration entry '`DocumentRoot`' followed by the path.

The next step is to setup REBOL. Just copy the file '`rebol031.exe`' to a directory on your local disk. Usually this will be '`C:/rebol/`'.

The next step is to copy the Biloba files to their destination. The directory '`w32setup/biloba/`' on the accompanying CD contains all required files. These must be copied to the directory you specified as DocumentRoot in the '`httpd.conf`' file. The location of the '`biloba.r`' file should be '`YOURDOCROOTHERE/biloba/biloba.r`'.

Edit '`biloba.r`' so that the very first line contains the location of the REBOL interpreter. Figure 2.3 illustrates how this line must look like in order to be interpreted by the Apache HTTP Server.

The setup is now complete — Start the Apache HTTP Server and visit `http://localhost/biloba/biloba.r?test.stx` with your browser!

2.2 Stand-alone Installation

This section explains how to setup Biloba to be used off-line. The only things you require are an interpreter for the REBOL programming language and the Biloba program files. Both are located in '`w32setup/`' on the accompanying CD.

Figure 2.3: Shebang line for biloba.r

```
#!c:/rebol/rebol031.exe -cs
```

Figure 2.4: change-dir Path For stxify.r

```
;change this to the directory where
;%stxify.r is located
change-dir %/C/Program%20Files/biloba/
```

Should you require a REBOL interpreter for a non-Windows platform, you have to download if from http://www.rebol.com/platforms.shtml.

First, copy the REBOL interpreter to a directory of your choice on your local disk. Usually this will be '`C:/rebol/`'.

The next step is to copy the Biloba directory ('`w32setup/biloba/`') to a place on your local disk. Associate '`.r`' files with the REBOL interpreter by double clicking on '`stxify.r`' and selecting the REBOL binary from the list, if the REBOL installation has not already done the association for you.

The Biloba prototype requires you to edit the file '`stxify.r`'. At the beginning of the file you will find a line starting with '`change-dir`' followed by a path introduced with the percent sign. Change the path after the percent sign to the path in which '`stxify.r`' is located. Note that this path *must* use forward slashes, must end with a slash, and must be an absolute path starting at the root. Also note that blanks need to be encoded as '`%20`'. For a typical installation this line is shown in Figure 2.4.

Optionally you can assign the extension '`.stx`' with Biloba's command line interface '`stxify.r`'. To do this double click on '`welcome.stx`' and select '`stxify.r`' to be the default application for '`.stx`' files.

2.3 Using the Command Line Interface

Biloba can be used from the command line to prepare files offline in various formats. Figure 2.5 shows the command line switches that can be used with '`stxify.r`'.

If no output format is specified, you will be asked to enter one. Available output formats are *debug*, *xml*, *html*, and *tex*.

If no source files are specified, you will be asked to enter one.

A file with the same name as the input file but the extension changed to the name of the output format. Note that previous existing files with that name will be overwritten without prior warning!

Figure 2.5: Command Line Switches for stxify.r

```
Usage: stxify.r [options] file...
Options:
    -h            Display this information
    -o <format>   Specify output format of following input files
                  Permissible formats are: debug, xml, html, tex

Examples:
    stxify.r -o xml fileA.stx fileB.stx
```

Chapter 3

Development

3.1 Writing Extension Modules

The figure mechanism in Biloba is designed to be easily extensible. New syntax rules for figures can be added to the system just by adding modules that adhere to the rules outlined in this section.

3.1.1 What are Figure Modules?

Whenever a line in a document is spontaneously indented, that is indented without a prior heading, the line and all following lines with the same level of indentation or more will be extracted from the document and rendered as figure.

The most basic type of figure is a verbatim area. Everything entered will appear exactly as typed in the source. Physical line breaks as well as blanks are preserved — no formatting is applied.

This is the default figure type. Every figure without an explicit type parameter will be treated as verbatim area.

3.1.2 Parameters

If the first lines of a figure are introduced with a hash sign (#), they have a special meaning.

A line with only a hash sign and some text after it will be treated as the figure's title. If the text after the hash sign contains a colon ('`:`'), the line is treated as key/value pair with the key as the left side and the value the right side of the

Figure 3.1: A Sample Figure

```
#This is the title
#type:image
path/to/image.jpg
```

Figure 3.2: Document That Uses the TRANSFORM Figure

```
#Text converted to UPPERCASE
#type:transform
#case:uppercase
This TEXT will be
tRaNsFoRmEd to
UPPERCASE characters,
however useful this
may be ...
```

colon.

One such key, *type*, has a pre-defined meaning: it specifies the figure module that is called to process the parameters and the figure text.

All other key/value pairs have no pre-defined meaning but can be assigned one by developers of a figure module.

Figure 3.1 shows an example of a figure. The figure text will be parsed by the module '`image`' located in the directory '`modules/`'. This is the module that inserts images in your document.

3.1.3 A simple Figure Module

We will now create simple figure module called *TRANSFORM.* It takes the figure text and transforms all characters to lowercase. This behaviour can be influenced with the parameter '`case`', which allows the values 'lowercase', 'uppercase', and 'preserve'. A typical invocation of this figure module can be seen in Figure 3.2.

While this figure module is not particularly useful, it illustrates all the concepts required for creating a figure module.

In order to add a figure module, you have to create a file with the name of the module. In our case, we create a file called *transform* with no extension and put it into the directory '`modules/`' where all figure modules must be located.

Figure modules are incepted as functions in the REBOL programming language. The function has to accept two arguments

Figure 3.3: Source Code for the TRANSFORM Figure Module

```
func [ headers lines /local transformed-string ][
  ;create a string that contains the first line
  transformed-string: copy lines/1

  ;so we can easily append the other lines after
  ;inserting a newline character
  lines: next lines ;skip first, already added, line
  forall lines [
    insert tail transformed-string newline ;insert newline
    insert tail transformed-string lines/1 ;insert the line
  ]

  either find headers "case:uppercase" [
    ;transform to uppercase
    uppercase transformed-string
  ][
    if not find headers "case:preserve" [
      ;transform to lowercase
      lowercase transformed-string
    ]
  ]

  ;return a valid document node [ 'node-type [ "node's content" ] ]
  reduce [ 'verbatim reduce [transformed-string] ]
]
```

- a block containing all headers, and
- a block containing the lines

Figure 3.3 shows the source code of the TRANSFORM figure module. The first line starts the function definition with the first block containing the parameters '`headers`' and '`lines`'. These are the words through which the two blocks will be passed. The '`/local`' string is called a refinement. It specifies that '`transformed-string`' is a local variable.

'`either find headers "case:uppercase"`' tests if '`"case:uppercase"`' was passed as header. If so, the function transforms the text to uppercase characters using the REBOL function '`uppercase`'.

Finally the function has to return a valid document node which consists of a document node identifier and a block of further nodes and strings that make up the node's content. These values are enclosed in a REBOL block.

To create more sophisticated modules, you need to have a background in the REBOL programming language. The REBOL Reference Manual is available online at http://www.rebol.com/users/valurl.html.

Adding an Output Writer

Chapter 4

Expert Formatting Commands

4.1 Escaping

Sometimes you want to prevent Biloba from interpreting characters and have them preserved just as they are. A frequent example is the use of a colon (:) inside a figure caption. Naively writing the colon inside the caption will turn the caption into a key/value pair.

Fortunately there is a mechanism to prevent this. By adding a backslash in front of any character, the character will appear exactly the same way in the output. This can be used to prevent the colon from becoming a key/value pair delimiter. Figure 4.1 shows an example.

Note that this can be applied to any character you feel necessary.

4.2 Empty Paragraph

The empty paragraph, a single period as the only character in a line, can be used to directly influence the current level of indentation without the need for text.

This is especially useful if you want to have a figure immediately following another figure. Without the empty paragraph the second figure would not be detected as a second figure but the will be joined to a single, larger figure. Without the empty

Figure 4.1: Escaping: Use Backslash to Prevent Colon From Becoming a Delimiter

```
#Escaping: Use Backslash to Prevent Colon From Becoming a Delimiter
...
```

Figure 4.2: Two Adjacent Figures

```
  #Image 1
  #type:image
  image1.jpg
.
  #Image 2
  #type:image
  image2.jpg
```

paragraph you would have to add some text between the figures for clarity, which is sometimes not an option.

An example (see Fig. 4.2) will clarify this.

4.3 Comments

Comments are used to prevent Biloba from parsing a single line or a group of lines.

Lines starting with a hash sign (‘`#`’)in the very first column will be ignored.

To ignore a block of lines, enclose the lines between ‘`#=ignore`’ and ‘`#=end`’, both of which must appear on a line of its own and the hash sign needs to be in the very first column.

4.4 Preserving Input

Text between ‘`#=preserve`’ and ‘`#=end`’ will be transferred 1:1 to the output document.

Note that this feature limits the output format independence. Text inside a preserve block can violate rules in the output format. Biloba can no longer ensure that the created markup conforms the the rules of the output format.

Use this only if you know the document will not be transformed into other formats than the one you wrote the preserve blocks for.