Usage notes for Windows Browser 2.9

Hippocrates Sendoukas
December 1, 1994

Topics

Introduction

This document summarizes the use of WBR which is a browser for text files under MS-Windows (3.1 and later) and Windows NT. All program functions can be accessed through standard menus or accelerator keys that are shown after each menu entry. The program comforms to the MDI specification. Its operation is quite simple; it reads one or more text files into memory and displays them on the screen. You can move around the file by using the scroll bars or the arrow keys. You can also use regular expressions for convenient searching. Finally, you can use any installed font to display your files. All options as well as window position and size are recorded in the file "wbr.ini" (or "wbr2.ini" for the NT version) for your convenience. The browser also records the last eight files that you opened and appends their names to the "File" submenu. You can open any of these files quickly by clicking on the desired name.

Usage

The program accepts several command line switches and filenames. The purpose of these command line arguments is to simplify the use of the browser from other programs; I chose this method of interprocess communication since most programs can execute system commands, while relatively few support more advanced forms of communication such as DDE or registered messages. The formal usage is:

wbr [-1] [-x] [-l] [-c] [-n] [-w] [-s xxxx] [-g NN] [-s xxxx] [filename ....]

where items in square brackets are optional. If you specify one or more filenames upon invocation, the browser will open all requested files in separate MDI windows.

The "-1" switch forces a single instance of the program; if you call the browser twice with this switch, the second instance will only instruct the first instance to open (or refresh) the requested files. This is a particularly useful feature since the calling program does not need to check if the browser is already running. All other command line switches are meant to be used together with this switch. When you use this switch, the program does not append any filenames to the "File" submenu; this is useful for the temporary log files of the dvi driver.

The "-x" switch instructs the browser to display the requested file in a maximized window.

The "-l" switch is equivalent to the combination of the "-1" and "-x" switches; its only purpose is backward compatibility with previous versions of the program.

The "-c" switch instructs the browser to close any other instance of itself. This is useful if you want to terminate the program automatically.

The "-n" switch instructs the browser to avoid refreshing any files that are already open.

The "-w" switch instructs the browser to avoid issuing warnings about missing files.

The "-s xxxx" switch instructs the browser to search for the string "xxxx" in the active window.

The "-g NN" switch instructs the browser to go to line NN in the active window. If NN is negative, it indicates a line relative to the end of the file.

The "-m xxxx" switch instructs the browser to read all its strings from the file "xxxx" instead of the default file "wbr.str".

User defined strings

All strings of the program (messages, menu entries, etc.) reside in the file "wbr.str" which is a plain ASCII file; this lets you modify all strings and optionally translate them to various languages. The string file should be located either in the current directory or someplace in your path; the browser reads it upon startup. The format of the string file is very simple: it is composed of strings that are separated by zero or more spaces, tabs or newlines. Each string is delimited by double quotes (") and the backslash character (\) is an "escape" character as in the C language; the program recognizes the following escape sequences:

\a	Introduces the ASCII bell character (7)
\b	Introduces a backspace (ASCII 8)
\f	Introduces a formfeed (ASCII 12)
\n	Introduces a newline (ASCII 10)
\r	Introduces a carriage return (ASCII 13)
\t	Introduces a tab (ASCII 9)
\xNN	Introduces a character whose code is equal to NN, where NN are two hexadecimal digits. NN cannot be equal to "00".

Any other character after a backslash is accepted literally (ie., it is included in the string); for example, the sequence \\ introduces a single backslash into the string and the sequence \" introduces a double quote. The only exception is the sequence \<Actual Newline> which simply continues the string to the next line; this can be convenient for long strings.

The string file can also include comments: each comment is introduced by a percent character (%) and continues to the end of the line.

Some of the strings are used as templates for the "printf" function, so you may encounter a percent character followed by various characters. These sequences are interpreted as following:

%s	will be replaced by a string at runtime
%u	will be replaced by an unsigned integer at runtime
%d	will be replaced by a signed integer at runtime
%ld	will be replaced by a long signed integer at runtime
%g	will be replaced by a floating point number at runtime
%%	will be replaced by a single percent character at runtime

If you do modify strings that contain any of the above special sequences, MAKE SURE that you do not introduce any new specifiers or change the order or type of existing ones; the only valid modification is to delete one or more specifiers provided that they are the last ones in the string. The program checks the validity of each string and will complain if any of the above rules

are violated. Each string can contain up to 4096 characters. There is no limit on the total number of characters in the string file. The program assumes that the string file uses the ANSI (Windows) character set as opposed to the OEM (DOS) character set; this is relevant only for characters whose code is greater than 127. If you do use such characters, make sure that you edit the file with a Windows editor instead of a DOS editor.

Some of the strings may be used as menu entries; these strings often contain an ampersand (&) which indicates that the following character should be underlined. The sequence && introduces a single ampersand in the menu entry.

As mentioned above, the program reads the strings from the file "wbr.str" upon startup. One can override this default string file by using the command line switch "-m". If for example you execute the command:

<pre>   wbr -m special.str</pre>

the program will read the strings from the file "special.str" instead of "wbr.str". This facilitates the switching among various versions of the string file.

The program comes with several versions of the string file for a few languages. The base file "wbr.str" contains comments about the meaning of each string; these comments should be helpful when adapting the string file to another language. If anybody translates the strings to his native language and is willing to share it with the rest of the program's users, I will be happy to distribute the string file with the upcoming versions of the program.

Searching for text

The browser supports searches using "regular expressions". This is a very powerful method that lets you perform "approximate" searches. Regular expressions are patterns that can match more than one string. They are composed of normal and special characters. In their simplest form they contain only normal characters and can match only a single string: for example the expression "abc" will match the substring whether it occurs by itself or within a word. It will match the strings "abc", "abcd", "0abcd"; it will not match the string "Abc" or "aBCd". Such simple patterns are useful, but there are many cases where one needs something more powerful; suppose for example that we want to find the keyword "if" in a Pascal program: Pascal is not case sensitive, so we have to check all possible spelling combinations; furthermore, we do not want the pattern to occur within other words. Regular expressions provide many such capabilities which are controlled by special characters embedded within the pattern. These characters and their functions are:

Character	Function
.	Matches any character
[...]	Matches a character from those inside the bracket
*	Matches zero or more instances of the preceding character
+	Matches one or more instances of the preceding character
?	Matches zero or one instances of the preceding character
{n1,n2}	Matches n1 to n2 instances of the preceding character
^	Matches the beginning of a line
$	Matches the end of a line
<...>	Matches an entire word
(...)	Brackets a regular expression

The first group of special characters involves the specification of single characters. A dot (.) denotes any character; for example, the pattern "i." matches the strings "if", "in" or "i7". A set of brackets enclosing some characters denotes a character class and the expression matches any single character from the character class; the pattern "[Ii]f" matches the strings "if" and "If". You can also abbreviate the character class by using a dash: the pattern "[a-z]" will match any lowercase letter. You can also specify a negative character class, where the expression will match any character except those in the character class; this is accomplished by specifying a caret sign (^) immediately after the left bracket; the pattern "[^a-z]" will match any character except a lowercase letter.

There are also some special characters dealing with the number of instances of a character (this is called closure). An asterisk denotes zero or more occurrences of the preceding character; the pattern "go*d" matches the strings "gd", "god", "good", "goood", etc. A plus sign denotes one or more instances of the preceding character; the pattern "go+d" matches the strings "god", "good", "goood", but not the string "gd". A question mark matches zero or one instances of the preceding character; therefore, the pattern "go?d" will match only the strings "gd" and "god". The special characters {n1,n2} match n1 to n2 instances of the preceding character; the pattern "go{2,3}d" matches the strings "good" and "goood", but not the strings "god" or "gooood". There are also two variants of the last special characters. The special characters "{n1}" match exactly n1 instances of the preceding character, while the special characters "{n1,}" match at least n1 instances of the preceding character.

The third group of special characters deals with the location of the string. If the first character of the regular expression is a caret (^), the expression will be matched only at the beginning of a line. For example, the pattern "^abc" matches the string "abc" only if it occurs at the beginning of a line. Similarly the dollar sign ($) matches the end of a line; the pattern "abc$" matches the string "abc" only if it occurs at the end of a line. The pattern "^$" will match all empty lines. Note that these two characters are treated as special characters only if they occur in the beginning or the end of the regular expression. That is, the pattern "a$b^c" will match the string "a$b^c" regardless of its position on a line. Another set of special characters are "<" and ">". These facilitate the matching of entire words, ignoring any matching substrings embedded in other strings. For example, the pattern "<abc>" will match the string "abc" by itself, but not the string "abcd".

Sometimes we need to specify parts of the regular expression. For this reason, we use the special characters "(" and ")" to bracket a part of the expression. The matching behavior is not affected, but we can refer to these parts of the expression by the notation "\N" where N is a digit between 1 and 9. Suppose that we want to find all sequences of two identical characters; this would appear quite difficult since we do not know these characters in advance. Regular expression bracketing solves this problem quite elegantly: the pattern "(.)\1" will find the desired characters.

Since the characters . [ ] * + ? ^ $ ( ) { } have a special meaning in the context of regular expressions, we need another special notation when we want to search for them literally: in order to suppress the special meaning of any character ("quote" it), we can precede it by a backslash. Therefore if we want to find an asterisk, the search string will be "*"; if we want to find a left parenthesis, the search string will be "\(". To find a backslash by itself, the search string will be "\\". For the user's convenience, the program also accepts some standard escape sequences. The sequence "\n" indicates the newline character (^J), "\r" indicates the carriage return character (^M), "\t" indicates the tab character (^I), "\b" indicates the backspace character (^H),

"\a" indicates the bell character (^G), "\f" indicates the formfeed character (^L), and "\xNN" indicates the character with code NN where NN are two hexadecimal digits. These escape sequences can be used anywhere in the search patterns.

Another fundamental rule is that regular expressions try to match as many characters as possible. While this is usually desirable, there are cases where it can be surprising. Suppose for example that the search pattern is "\(.*\)" (the parentheses are quoted to indicate that we want to match them literally), and our text is "(one) and (two)". In this case, the pattern will match the entire line, since it will find the last right parenthesis at the end. If we wanted only the first pair of parentheses, we should specify "\([^)]*\)" as our search pattern. In this way, it will match the substring "(one)". The right parenthesis inside the square brackets does not need to be quoted: almost all characters in a character class are taken literally.

Another point that we should keep in mind is that the matching of closures can be sometimes surprising. Suppose for example that the search string is "s*". Since the search string specifies zero or more occurrences of the letter s, it will match anything, including the empty string. The problem is that the search string consists only of a closure, and therefore it matches anything. A simple solution to this problem, is to avoid using such search strings.

For a more thorough discussion of regular expressions, one can read any book describing the operation of the Unix "ed", "sed", "grep", "awk" or "lex" commands. In general, regular expressions are quite powerful means of manipulating text. They have however several limitations: for example, the construction of some compound patterns can be complicated. Furthermore, they are line oriented: that is, they cannot match expressions spanning more than one line.

Regular Expression Errors

It should be obvious by now that not all regular expressions are valid. The program checks each expression and complains if it is illegal. The possible error messages are:

"**Invalid number**": This means that the program did not find an expected number. This can happen when you do not specify a number incide a pair of braces.

"**Invalid subexpression number**": This means that you specified an undefined subexpression. The expression "(.)\1" is valid because "\1" corresponds to the subexpression "(.)". However, the expression "(.)\2" is invalid because you did not specify two subexpressions (using parentheses).

"**Unbalanced parentheses**": This indicates that the left and right parentheses for bracketing subexpressions are unbalanced.

"**Too many (**": This means that you tried to bracket more than 9 subexpressions.

"**More than 2 numbers in {   }**": You can specify one or two numbers within a pair of braces. This message indicates a violation of this rule.

"**} expected**": This message arises from an ill-formed pair of braces. The only valid tokens inside the braces are numbers or a comma after the first number.

"**First number exceeds second in {   }**": Two numbers inside a pair of braces indicate the minimum and maximum number of times that the previous character should be matched. Obviously, the minimum cannot be more than the maximum.

"**Unbalanced square brackets**": This message indicates that the left and right brackets for a character class do not match.

"**Expression too complicated**": The program checks a regular expression for errors and "compiles" it to an intermediate form for quick searching. This error can arise if the intermediate form is too large, which typically happens if you specify too many character classes (above 15).

"**Invalid hexadecimal number**": As mentioned previously, the program lets you specify any 8-bit character by using a hexadecimal notation "\xNN" where NN is a valid hexadecimal number. All such numbers should consist of two hexadecimal digits exactly: that is, use \x08 instead of \x8. This error occurs if any of the two characters following \x is not a valid hexadecimal character.

"**Unknown error**": This is a "catch all" that should never occur. If you see it, you can be sure that there is a bug in the program and I would appreciate if you let me know about it, so I can fix it.

Selecting a font

The program lets you use any installed font in your system; it can deal with variable pitched, bitmap, vector, TrueType or ATM fonts. When you select the "Font" entry in the "Options" menu, you are presented with a dialog box containing all the installed fonts at all sizes and styles. Your selection is automatically recorded in "wbr.ini" (or "wbr2.ini" for the NT version), so you do not need to set it more than once.

Copying text to the clipboard

You can copy the entire text of the active file to the clipboard by selecting the "Copy" command from the "Edit" menu. There is no way in this version to copy only a portion of the text.

Packing List

Make sure that you have all the relevant files:

Filename	Description
wbr.exe	Text file browser
wbr.hlp	Help file using standard fonts
wbr2.hlp	Help file using larger fonts
wbr.wri	Printable documentation
wbr.str	String file
miscwin.dll	Utility routines
ctl3dv2.dll	More utility routines
wbr2.exe	Windows NT version of wbr.exe
miscwin2.dll	Windows NT version of miscwin.dll
ctl3d32.dll	Windows NT version of ctl3dv2.dll

The only required files for the browser are "wbr.exe", "miscwin.dll" and "wbr.str". The files "ctl3dv2.dll" and "ctl3d32.dll" are distributed by Microsoft Co. which requires them to reside in your "\windows\system" directory. All other files except for the documentation must reside either in a directory specified by the PATH environment variable, or the base directory of Windows. If you use a resolution of 1024x768 or higher, you may find the standard help fonts a bit hard to read. In that case, rename the file "wbr2.hlp" to "wbr.hlp" so that the browser uses this file instead of the other one.

Caveats

The program reads files into memory and just displays them; it will complain if there is not enough memory to load a file (which is unlikely since Windows has virtual memory). In general, it can handle quite large files (unlike Notepad) without excessive demands on memory. The program has proven quite useful to me, and I hope that it serves you equally well. I have tested and debugged it extensively, but I cannot afford to make any guarantees; anybody who uses it assumes all risks. On the other hand, if you find any bug or have any suggestion for improvements, I will be more than happy to hear about it and I will do my best to fix it. You can contact me via e-mail at "isendo@leon.nrcps.ariadne-t.gr" or regular mail at "3 Sifnou St., Athens 11254, Greece".

Licensing Agreement

The author of this software grants to any individual or non-commercial organization the right to use and to make an unlimited number of copies of this software. Commercial entities may use the software for an evaluation period of two weeks; any further use requires a license from the author. You may not decompile, disassemble, reverse engineer, or modify the software. This includes, but is not limited to modifying/changing any icons, menus, or displays associated with the software. This software cannot be sold without written authorization from the author. This restriction is not intended to apply to connect time charges, or flat rate connection/ download fees for electronic bulletin board services. The author of this program accepts no responsibility for damages resulting from the use of this software and makes no warranty or representation, either express or implied, including but not limited to, any implied warranty of merchantability or fitness for a particular purpose. This software is provided as is, and you, its user, assume all risks when using it.

Hippocrates Sendoukas

E-mail address: isendo@leon.nrcps.ariadne-t.gr
Mail address: 3 Sifnou St., Athens 11254, Greece