



# Oolite Regular Expression Library

Version 1.0 Beta2 December 8 1995

Ported by Rob Reiss (Rob\_Reiss@msn.com)

## Introduction

This package provides C++ and Visual Basic functions that perform regular expression matching. The library was specifically built for Windows95 and Windows NT operating systems and has not been tested under any other platforms. Please see the Legal Notice for information on redistribution of this package and a DISCLAIMER OF ALL WARRANTIES.

## Acknowledgments

This library was created by modifying regular expression code originally written by Henry Spencer. I found Henry Spencer's code among the source files for the TCL language created by John Ousterhout. I would like to thank both of these people for the great source code they make available to the world.

### General:

[Installation](#)

[Regular Expressions](#)

[Sample Code](#)

[Legal Notice](#)

### Visual Basic API

[RegExp](#)

[RegSub](#)

[RegExpIndx](#)

### C++ API

[OolRegExpr](#)

[OolRegIndx](#)

### C++ Low Level API

[OolRegComp](#)

[OolRegCompCache](#)

[OolRegExec](#)

[OolRegRange](#)

## File Info:

[oolregex.dll](#), [oolregex.h](#), [oolregex.lib](#), [oolregex.bas](#), [VBTest](#), [CPPTTest](#)

## Source Code And DLL License:

Source code and a royalty free licenses to distribute the oolregex.dll separately is available for \$500 (Dec 8 1995, price may change) Note: you are still allowed to distribute this package and use it for any reason without charge on the condition that you do not break it up (i.e. distribute the oolregex.dll with out the rest of the package). See the Legal Notice. For more information please email Rob\_Reiss@msn.com

P.S. You can also obtain the core regular expression code for free through the TCL distribution. Please see <http://www.sml.com/research/tcl> for more information about TCL.

**oolregex.dll:**

Windows 32bit dll that contains the code to perform regular expression matching on strings. This file should be installed into the end users Windows System directory.

**oolregex.h:**

C++ header file that can be included in your C++ projects to access the regular expression calls in oolregex.dll.

**oolregex.lib:**

C++ link library that can be linked to your C++ projects to link to oolregex.dll

**regexp.bas:**

Visual Basic 4.0 module that can be included in VB applications to access the regular expression library.

**VBTest:**

A Simple example program showing how to access the regular expression library from Visual Basic

## **CPPTest**

A Simple example program showing how to access the regular expression library from C++

## Introduction

The regular expression library can be used to match regular expressions to strings from Visual Basic or C++. This library was created by modifying code distributed with the TCL language. The regular expression code was originally written by Henry Spencer. John Ousterhout incorporated Henry Spencer's code into TCL where I found it. I modified the code so that it was easily called by C++ and Visual Basic. I also changed the error reporting scheme and made it thread safe. Please see the Legal Notice for information on usage and redistribution of this package, and for a DISCLAIMER OF ALL WARRANTIES.

Enjoy,  
Rob Reiss  
Rob\_Reiss@msn.com



## **Installation**

To install Oolite Regular Expression library simply copy the file oolregex.dll to your Windows directory. (NOTE: you do not need to do this to use the sample programs)

## Sample Code

Included with the RegExp Library are the two programs VBTest and CPPTTest. These programs demonstrate how to use the RegExp Library from Visual Basic and C++ respectively. I suggest you jump right in and look at the code in these examples to get a quick understanding of how to use the regular expression library.

The main functions to look at are Match\_Click in VB and CRegExpTestDlg::OnButtonMatch in C++.

CPPTTest was built using Microsoft Visual C++ 4.0 and the source code is located in the CPPTTest directory.  
VBTest was built using Microsoft Visual Basic 4.0 and the source code is located in the VBTest directory.

```

Private Sub Match_Click()
    Dim Match As Boolean
    Dim str1 As String, str2 As String, str3 As String, str4 As String
    On Error GoTo ErrorHandler

    Match = RegExp(inputString.Text, RegExpInput.Text, str1, str2, str3)
    Match = RegSub(inputString.Text, RegExpInput.Text, Substitute.Text, str4, 1)
    res1.Text = str1
    res2.Text = str2
    res3.Text = str3
    res4.Text = str4
    If Match = True Then
        MsgBox "String Matches", vbOKOnly, "Match Result"
    Else
        MsgBox "String does not match.", vbOKOnly, "Match Result"
    End If
    Exit Sub
ErrorHandler:
    If Err.Number <> 0 Then
        Msg = Err.Description
        MsgBox Msg, , "Error", App.helpfile, Err.HelpContext
    End If
End Sub

```

```
void CRegExpTestDlg::OnButtonMatch()
{
    UpdateData(TRUE);
    int match;
    char* res1;
    char* res2;
    char* res3;
    match = OolRegExpr(m_string, m_regexp, &res1, &res2, &res3, NULL);
    m_res1 = res1;
    m_res2 = res2;
    m_res3 = res3;
    UpdateData(FALSE);
    if (match == S_OK)
    {
        AfxMessageBox("Pattern Matches!");
    } else {
        AfxMessageBox("No Match");
    }
}
```

## Error Codes

E_UNEXPECTED	"Unexpected Error"
E_OUTOFMEMORY	"Out of Memory"
E_INVALIDARG	"Invalid Argument"
E_REGEXPTOOBIG	"The Regular Expression was too big"
E_REGEXPTOOMANYPAREN	"Too many ()'s in the Regular Expression"
E_REGEXPUNMATCHPAREN	"Unmatched ()'s in the Regular Expression"
E_REGEXPSTARPLUSEMPT	"Possible problem with *+"
E_REGEXPNESTED	"Nested *?+ in Regular Expression"
E_REGEXPINVALIDBRKRANGE	"Invalid [] range in Regular Expression"
E_REGEXPUNMATCHBRACKET	"Unmatched [] in Regular Expression"
E_REGEXPPOPFOLLOWSNOTHING	"?+* follows nothing in Regular Expression"
E_REGEXPTRAILINGSLASHS	"Trailing backslashes in Regular Expression"

## Visual Basic RegExp Function

Matches a regular expression to a string returning TRUE if there is a match and FALSE otherwise. RegExp also returns the substrings of the input string that matched the sub-expressions in the regular expression.

### Syntax

**RegExp**(*String*, *Exp* [, *MatchVar*, *subMatchvar*, ...])

Argument	Description
<b>String</b>	Required. A string expression containing the input string to be scanned.
<b>Exp</b>	Required. A string expression containing the regular expression.
<b>MatchVar</b>	Optional. A string variable that will be set to the range of <i>String</i> that matched all of <i>Exp</i>
<b>subMatchVar</b>	Optional. A string variable that will be set to the range of <i>String</i> that matched the nth subexpression within <i>Exp</i> .

### Returns:

**True** if Exp matches part or all of String, otherwise **False**.

### Remarks:

Determines whether the regular expression *Exp* matches part or all of *String* and returns True if it does, False if it doesn't.

If additional arguments are specified after *String* then they are treated as the names of variables in which to return information about which part(s) of *String* matched *exp*. *MatchVar* will be set to the range of *String* that matched all of *exp*. The first *subMatchVar* will contain the characters in *String* that matched the leftmost parenthesized subexpression within *exp*, the next *subMatchVar* will contain the characters that matched the next parenthesized subexpression to the right in *exp*, and so on.

**Errors:** See [Error Codes](#)

### Example:

```
Dim substr1 As String, substr2 As String
Dim match As Boolean
match = RegExp("abc", "(ab|a)(b*)c", substr1, substr2)
```

After these calls the variables will hold the following values:

match = TRUE, substr1 = "abc", substr2 = "ab"

### See Also

[RegSub](#), [RegExpIndx](#)

## Visual Basic RegSub Function

---

Perform a string substitution based on regular expression pattern matching

### Syntax

**RegSub**(*InputString*, *Exp*, *SubstituteString*, *ReturnString*, [*Position*])

Argument	Description
<b><i>InputString</i></b>	Required. A string expression containing the input string to be scanned.
<b><i>Exp</i></b>	Required. A string expression containing the regular expression.
<b><i>SubstituteString</i></b>	Required. A string expression that will be used to replace the portion of <i>InputString</i> that matches <i>Exp</i>
<b><i>ReturnString</i></b>	Required. A string variable that will be set to the result of substituting in <i>SubstituteString</i> into <i>InputString</i> .
<b><i>Position</i></b>	Optional: An Integer specifying which sub-expression should be used for substitutions

### Returns:

**True** if *Exp* matches part or all of *String*, otherwise **False**.

### Remarks:

This command matches the regular expression *Exp* against *InputString*, and it copies *InputString* to the variable whose name is given by *ReturnString*. If there is a match, then while copying *InputString* to *ReturnString* the portion of *InputString* that matched *Exp* is replaced with *SubstituteString*. If *Position* is given then the substring that matched the *nth* sub-expression will be replaced by *SubstituteString*, where *nth* is given by *Position*.

**Errors:** See [Error Codes](#)

### Example:

```
Dim substr1 As String
Dim match As Boolean
match = RegSub("abc", "(ab|a)(b*)c", "Hello", substr1, 1)
```

After these calls the variables will hold the following values:

match = TRUE, substr1 = "Helloc"

### See Also

[RegExp](#), [RegExpIndx](#)

## Visual Basic RegExplndx Function

---

### Syntax

**RegExplndx**(*InputString*, *Exp* , *IndexArray*, *NoCase*)

Argument	Description
<b><i>InputString</i></b>	Required. A string expression containing the input string to be scanned.
<b><i>Exp</i></b>	Required. A string expression containing the regular expression.
<b><i>IndexArray</i></b>	Required. A (2,x) dimension array of Integers. RegExplndx will return the character indexes of the substrings that <i>Exp</i> matches in <i>String</i> in <i>IndexArray</i>
<b><i>NoCase</i></b>	Required. Can be set to 0 or 1. When <i>NoCase</i> = 1 the regular expression comparison will use case <u>insensitive</u> matching. . When <i>NoCase</i> = 0 the regular expression comparison will use case <u>sensitive</u> matching.

### Returns:

An HRESULT with a value of  
**S\_OK** if the expression matched  
**S\_FALSE** if the expression didn't match  
An Error Code if an error occurred.

### Remarks:

Determines whether the regular expression *Exp* matches part or all of *InputString* and returns S\_OK if it does, S\_FALSE if it doesn't.

After the call to **RegExplndx** *IndexArray* will hold the starting and ending positions of the substrings that matched *Exp*. i.e.

Indx(0,0) = Position of first character of substring of *InputString* that matched *Exp*

Indx(1,0) = Position of last character of substring of *InputString* that matched *Exp*

Indx(0,n) = Position of first character of substring of *InputString* that matched the nth sub-expression of *Exp*

Indx(1,n) = Position of last character of substring of *InputString* that matched the nth sub-expression of *Exp*

---

### See Also

RegExp, RegExplndx

---

## C++ OolRegIndx Function

### Syntax

HRESULT OolRegExpIndx(const char\* *InputString*,const char\* *Exp* , *IndexArr*(2,NSUBEXP), *NoCase*)

Argument	Description
<i>InputString</i>	Required. A string expression containing the input string to be scanned.
<i>Exp</i>	Required. A string expression containing the regular expression.
<i>IndexArray</i>	Required. A (2,50) dimension array of Integers. OolRegIndx will return the character indexes of the substrings that <i>Exp</i> matches in <i>InputString</i> in <i>IndexArray</i>
<i>NoCase</i>	Required. Can be set to 0 or 1. When <i>NoCase</i> = 1 the regular expression comparison will use case <u>insensitive</u> matching. . When <i>NoCase</i> = 0 the regular expression comparison will use case <u>sensitive</u> matching.

### Returns:

An HRESULT with a value of  
**S\_OK** if the expression matched  
**S\_FALSE** if the expression didn't match  
An Error Code if an error occurred.

### Remarks:

Determines whether the regular expression *Exp* matches part or all of *InputString* and returns True if it does, False if it doesn't.

After the call *IndexArray* will hold the starting and ending positions of the substrings that matched *Exp*. i.e.

Indx(0,0) = Position of first character of substring of *InputString* that matched *Exp*

Indx(1,0) = Position of last character of substring of *InputString* that matched *Exp*

Indx(0,n) = Position of first character of substring of *InputString* that matched the nth sub-expression of *Exp*

Indx(1,n) = Position of last character of substring of *InputString* that matched the nth sub-expression of *Exp*



## C++ OolRegExpr Function

---

Matches a regular expression to a string returning S\_OK if there is a match and S\_FALSE otherwise. OolRegExpr also returns the substrings of the input string that matched the sub-expressions in the regular expression.

### Syntax

HRESULT **OolRegExpr**(const char\* *String*, const char\* *Exp*, char\*...)

Argument	Description
<i>String</i>	string containing the input string to be scanned.
<i>Exp</i>	string containing the regular expression.
<i>Optional Args</i>	Optional. Any number of char* followed by NULL

### Returns:

**S\_OK** if Exp matches part or all of String, otherwise **S\_False**.

If an error occurs OolRegExpr will return an Error Code.

### Remarks:

Determines whether the regular expression *Exp* matches part or all of *String* and returns True if it does, False if it doesn't.

If additional arguments are specified after *String* then they are treated as the names of variables in which to return information about which part(s) of *String* matched *exp*. *MatchVar* will be set to the range of *String* that matched all of *exp*. The first *subMatchVar* will contain the characters in *String* that matched the leftmost parenthesized subexpression within *exp*, the next *subMatchVar* will contain the characters that matched the next parenthesized subexpression to the right in *exp*, and so on.

### Example:

```
char* substr1, substr2;  
HRESULT match = OolRegExpr("abc", "(ab|a)(b*)c", substr1, substr2, NULL)
```

After these calls the variables will hold the following values:

```
match = S_OK, substr1 = "abc", substr2 = "ab"
```

### See Also

OolRegIndx

---

## C++ OolRegComp and OolRegCompCache Functions

---

Compiles a regular expression returning a oolregexp structure that may be used in a call to OolRegExec, OolRegSub, and OolRegRange.

### Syntax

HRESULT **OolRegComp**(const char\* *Exp*, oolregexp\* *Reg*)

HRESULT **OolRegCompCache**(const char\* *Exp*, oolregexp\* *Reg*)

Argument	Description
<i>Exp</i>	string containing the regular expression.
<i>Reg</i>	The compiled regular expression will be passed back in this pointer

### Returns:

**S\_OK** if compilation of *Exp* worked, otherwise an Error Code.

### Remarks:

**OolRegComp** compiles a regular expression so that it can be used to match against strings using **OolRegExec**.

**OolRegCompCache** does the same thing as **OolRegComp** except that it uses a cache of compiled regular expression to speed up processing

WARNING: The caller is responsible for freeing the memory allocated for reg when you use

**OolRegComp** but the caller should NOT free *Reg* when **OolRegCompCache** is used.

(**OolRegCompCache** will automatically free storage when the cache becomes full)

### Example:

```
oolregexp* reg, regCached
HRESULT match = OolRegComp("(ab|a)(b*)c", reg)
HRESULT match = OolRegCompCache("(ab|a)(b*)c", regCached)
---- use reg and regCache ----
free(reg); // Make sure to free reg when your done. BUT don't free regCached
```

### See Also

OolRegExec

## C++ OolRegExec Function

---

Matches a regular expression to a string returning S\_OK if there is a match, S\_FALSE if there is no match, and an Error Code otherwise.

### Syntax

HRESULT **OolRegExec**(oolregexp\* *Exp*, const char\* *String*, const char\* *StartString*)

---

Argument	Description
<i>String</i>	string containing the input string to be scanned.
<i>Exp</i>	oolregexp containing the compiled regular expression.
<i>StartString</i>	The starting point withing <i>String</i> to start the comparison

---

### Returns:

S\_OK if Exp matches part or all of String  
S\_False if Exp does not match any part of String  
Or an Error Code.

### Remarks:

Determines whether the regular expression *Exp* matches part or all of *String* and returns True if it does, and False if it doesn't.

### Example:

```
oolregexp* reg;  
char* str1 = "abc"  
HRESULT match = OolRegComp("(ab|a)(b*)c", reg)  
HRESULT match = OolRegExec(reg, str1, str1);  
free(reg);
```

After these calls the variables will hold the following values:  
match = S\_OK

### See Also

OolRegComp, Regular Expression

## C++ OolRegRange Function

---

Returns information about what portion of an input string matched.

### Syntax

HRESULT **OolRegRange**(oolregexp\* *Exp*, int *Index*, char\*\* *StartPtr*, char\*\* *EndPtr*)

Argument	Description
<i>Exp</i>	oolregexp containing the compiled regular expression.
<i>Index</i>	<i>Index</i> is used to indicate which subexpression in <i>Exp</i> you are interested in getting information about. ( <i>Index</i> = 0 for entire match)
<i>StartString</i>	returns a pointer to a pointer to the first character in the Input String that matched the <i>Index</i> 'th subexpression
<i>StartString</i>	returns a pointer to a pointer to the character just after the last character that matched the <i>Index</i> 'th subexpression

### Returns:

**S\_OK** if Exp matches part or all of String  
**S\_False** if Exp does not match any part of String  
Or an Error Code.

### Remarks:

Returns the pointers to the beginning and end of the portion of the input string that matched the *Index*'th subexpression

### See Also

OolRegComp, OolRegExec, Regular Expression

## REGULAR EXPRESSIONS (Taken directly from the TCL distribution.)

*Regular expressions are implemented using Henry Spencer's package (thanks, Henry!), and much of the description of regular expressions below is copied verbatim from his manual entry.*

A regular expression is zero or more *branches*, separated by "|". It matches anything that matches one of the branches.

A branch is zero or more *pieces*, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an *atom* possibly followed by "\*", "+", or "?".

An atom followed by "\*" matches a sequence of 0 or more matches of the atom.

An atom followed by "+" matches a sequence of 1 or more matches of the atom.

An atom followed by "?" matches a match of the atom, or the null string.

An atom is a regular expression in parentheses (matching a match for the regular expression), a *range* (see below), "." (matching any single character), "^" (matching the null string at the beginning of the input string), "\$" (matching the null string at the end of the input string), a "\e" followed by a single character (matching that character), or a single character with no other significance (matching that character).

A *range* is a sequence of characters enclosed in "["]. It normally matches any single character from the sequence. If the sequence begins with "^", it matches any single character *not* from the rest of the sequence. If two characters in the sequence are separated by "\-", this is shorthand for the full list of ASCII characters between them (e.g. "[0-9]" matches any decimal digit). To include a literal "]" in the sequence, make it the first character (following a possible "^"). To include a literal "\-", make it the first or last character.

## CHOOSING AMONG ALTERNATIVE MATCHES

In general there may be more than one way to match a regular expression to an input string. For example, consider the command

**regexp (a\*)b\* aabaaabb x y**

Considering only the rules given so far, **x** and **y** could end up with the values **aabb** and **aa**, **aaab** and **aaa**, **ab** and **a**, or any of several other combinations. To resolve this potential ambiguity **regexp** chooses among alternatives using the rule "first then longest". In other words, it considers the possible matches in order working from left to right across the input string and the pattern, and it attempts to match longer pieces of the input string before shorter ones. More specifically, the following rules apply in decreasing order of priority:

[1]

If a regular expression could match two different parts of an input string then it will match the one that begins earliest.

[2]

If a regular expression contains | operators then the leftmost matching sub-expression is chosen.

[3]

In \*, +, and ? constructs, longer matches are chosen in preference to shorter ones.

[4]

In sequences of expression components the components are considered from left to right.

In the example from above, **(a\*)b\*** matches **aab**: the **(a\*)** portion of the pattern is matched first and it consumes the leading **aa**; then the **b\*** portion of the pattern consumes the next **b**. Or, consider the following example:

**regexp (ab|a)(b\*)c abc x y z**

After this command **x** will be **abc**, **y** will be **ab**, and **z** will be an empty string. Rule 4 specifies that **(ab|a)** gets first shot at the input string and Rule 2 specifies that the **ab** sub-expression is checked before the **a** sub-expression. Thus the **b** has already been claimed before the **(b\*)** component is checked and **(b\*)** must match an empty string.

## KEYWORDS

match, regular expression, string

**Legal Notice:**

Copyright (c) 1995 by Rob Reiss  
All Rights Reserved

The "Oolite Regular Expression Library" may be freely distributed as a package. The contents of "the package" (compressed file) may not be broken up or added to, nor may any of the contents of the package be altered in any way without expressed permission from the author. The program is not public domain and remains the property of the author.

The source code for Oolite RegExp is available for \$500 please email [Rob\\_Reiss@msn.com](mailto:Rob_Reiss@msn.com)

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

