VB6 To Tcl mini−HOWTO

Mark Hubbard

Digital Connections Inc.

<markh@dcisite.com>

Pradeep Padala − Conversion from HTML to DocBook v4.1.

Revision History

Revision 1.0	2003−04−30	Revised by: tab
Initial release, reviewed by LDP		
Revision 0.9	2003−04−08	Revised by: ppadala
Docbook conversion		
Revision 0.8	2002−07−08	Revised by: mark
Original Document		

A 15 Minute Tcl Tour For Visual Basic and VBScript Programmers

Table of Contents

1. Introduction

VB and VBScript programmers: I know how you feel. Really. As a Microsoft Certified Professional in VB6, I've been doing those languages for 7 years. I really liked them, until I got over the hump in Tcl and started noticing the differences in flexibility that are shown here. If Tcl looks completely alien to you, and you wonder how in the world they dreamed it up, hold it up beside a piece of C code, or a UNIX shell script. I think those are what influenced it the most. UNIX shell scripts are a lot more advanced than MS Windows shell scripts, even those on NT/2000. In fact, UNIX shell scripts have a lot of the capabilities shown here. Both Tcl and shell script are based largely on string substitution. I chose to study Tcl over shell scripts because Tcl code is much more verbose and English−like (and therefore maintainable) than shell scripts, which tend to be cryptic. Some of the shell script command names are just punctuation alone!

Tcl also runs easily on the "big 4" PC platforms (Linux, *nix, Windows, Mac) as well as some others. This is promised by Java(tm), but delivered just as much (or more) by Tcl. And unlike Java and VB, Tcl is free of any commercial influences (which is true freedom, not just "free of charge"); over the years its development path sticks closer to what is really needed and wanted by you, its developers and potential developers. There has been no parent company to steer Tcl away from that and toward the company's own interests. The most startling contrast of all between Tcl and VB is that Tcl may even overshadow all the technical differences shown below.

---

2. Examples

Table 1. Differences

VB6	
Notes/differences	

`dim a as integer`
`dim b as integer`
`a=1 : b=0`
	`set a 1; se`

Separator for multiple commands per line. Tcl uses a semicolon. Multiple commands per line is generally considered b here.	
`' this is a whole line`	`# this is a`
Full−line comment. Neither language requires a space after the comment marker.	

`dim a as integer`
`a=1 'this is a partial−line comment`
	`set a 1 ;#`

Partial−line comment. Note the semicolon, used as if the comment is another command on that line.	

`dim s as string`
`s="/data/docs/vb6_to_tcl.htm"`
	`set s {/data`

Assignment of a string quoted with braces. Most Tcl substitutions are NOT done in a string quoted with braces. If the s be substituted at a later time. This is often done by commands that implement control structures, such as 'if' or 'while'. because it's important to getting 'good' in Tcl.	
`(No equivalent)`	`set s "/data`
Assignment of a quoted string. All Tcl substitutions (variables, commands, backslashes) are available within a quoted	
`(No equivalent)`	`set s /data`
Assignment of an unquoted string. All Tcl substitutions (variables, commands, backslashes) are available within an un (second argument to the set command). This works if there is no whitespace or certain other characters in the string. U	

`dim s as string`
`s = vbCrLf &"Free software is not just" &vbCrLf _`
`  &"about being 'free of charge'" &vbCrLf _`
`  &"but about freedom to create" &vbCrLf _`
`  &"and use the best possible tools." &vbCrLf`
	`set s {`
	`Free softwa`
	`about being`
	`but about fr`
	`and use the`
	`}`

Assignment of multi−line string. Note the more cluttered syntax in VB, which makes it more difficult to read than the	

```
dim s as string
dim t as string
s = trim(t)
```

```
set s [stri
```

Assignment of function return value. The third word of this set command is surrounded in square brackets. That means
command.

```
dim s as string
dim t as string
s = lcase(trim(t))
```

```
set s [stri
```

Assignment of function−of−function.

```
dim x as double
dim y as double
x = (y + 10) * 5
```

```
set x [expr
```

Assignment of result of a mathematical expression. The Tcl interpreter relies on the expr command to evaluate mathem
their implementation. When used explicitly, expr should be passed a single argument which is a string containing the e
add a certain increment to a variable. Try using the incr command for that instead.

```
dim s as string
s = s &"more text"
```

```
append s {m
```

Append to an existing string. This is one of the slowest operations in VB, but is typically very speedy in Tcl. Speed is

```
dim s as string
dim t as string
dim u as string
s = "I'll ask " & t & " to email " & trim(u) & " with the price"
```

```
set s "I'll
```

Building a string by substitution.

```
print "hello"
```

Displays hello

Print to console (VB actually prints to a form or to the debug window).

```
sub my_sub (byval a as integer, byval b as string)

    debug.print "I'll ask " & b

end sub

function my_function (byval a as integer, _
        optional byval b as string = "Mark") _
        as string

    my_function = "I'll ask " & b

end function
```

```
proc my_sub

    puts "I

}

proc my_func

    return

}
```

Procedure definition. Note that VB uses a separate syntax for subs and for functions. Tcl uses the **proc** command to de
Its first argument is a Tcl list of the parameters of the new procedure. Its second argument is a large string containing t

operations, including all references to command names and variable names, as well as (by default) string data comparis reference to the variable `B` within `my_sub` (b was defined as lower case).

```
dim i as integer
if i < 0 then i = 0 else i = i − 1
```

```
if {$i < 0}

# alternate
if {$i < 0}

# another a
if {$i < 0}
    set i 0
} else {
    incr i
}
```

'if' conditional execution. The Tcl 'if' command ignores the optional keywords 'then' and 'else' if they are present. Since To avoid syntax errors, also enclose any non−trivial test expression in braces. That way substitutions (such as $i here)

```
dim i as integer
i = 1
while i < 2000
    i = i * 2
wend

'alternate form
i = 1
do while i < 2000
    i = i * 2
loop
```

```
set i 1
while {$i <
    set i [
}
```

'while' loop. This is similar to the Tcl 'if' command in that it takes a test expression as its first argument, followed by a

```
dim i as integer
for i = 0 to 8
    'nine passes 0−8
    debug.print i
next
```

```
for {set i
    # nine
    puts $i
}

# alternate
for {set i
    # again
    puts $i
}

# another a
for {set i
    # nine
    puts $i
}

# yet anoth
set i 1
for {} {[in
    # nine
    puts $i
}
```

'for' loop with an integer counter. In Tcl (or any other language) this is equivalent to a 'while' loop. In some languages the initialization code, the test−for−continuation expression, and the increment code. Those pieces are not restricted to

```
dim c as new collection
dim o as object
c.add "Mark"
c.add "Roy"
c.add "Brian"
for each o in c
    debug.print o
next
```

```
set c [list
foreach o $
    puts $o
}
```

Loop through items in a data structure. In Tcl, a list data structure is used. VB has no direct equivalent to that, but a co
operations due to the overhead of using method calls to objects. Also note that there are *far more powerful and creativ*

```
dim s as string
select case s
    case "John"
        debug.print "Mellencamp"
    case "Steve"
        debug.print "Tyler"
    case else
        debug.print "Unknown"
end select
```

```
switch −exa
    John {pu
    Steve {
    default
}
```

One−of−many execution. Note the Tcl version is case sensitive. In VB it often is not, depending on the 'option compar
required, as opposed to a pattern match or regular expression match (this has no bearing on case sensitivity). Also note

```
on error goto handler
debug.print a 'a is undeclared.
...
handler:
debug.print err.number, err.description
```

```
if [catch {
    puts $a
} my_err] {
    puts "e
    puts "s
    # these
    # by th
} else {
    puts {A
    # the e
}
```

Error handling. In VB, handling errors concisely can be a problem, especially if different actions need to be taken base
addition, Tcl automatically provides a stack trace of the code that failed. In VB, the stack trace has to be explicitly buil
This is an advantage for Tcl when debugging in the field. Note that **catch** returns a boolean 1 or 0, which is typically u

```
(No equivalent)
```

```
set i [expr
```

Pass an arbitrary mathematical expression to the interpreter for evaluation. This could be an expression entered by the
available at all in VB.

```
(No equivalent)
```

```
set s [eval
```

Pass arbitrary code to the interpreter for execution. This could be some script entered by the user, or composed by earl

```
(No equivalent)
```

```
source my_s
```

Pass an arbitrary filename to the interpreter for execution of that file as a script. This is one of the most powerful aspe

```
(No equivalent)
```

```
set var_nam
incr $var_na
```

Perform operations on an arbitrarily−chosen variable. The code shown here will increment the variable `marks_age`.
command are subject to one pass of substitution by the interpreter just prior to execution. So any part of any command
one of the most powerful aspects of Tcl. It is not available at all in VB.

```
dim s as string
dim li as string
dim f_num as integer
s = ""
f_num = freefile
open "my_file.txt" for input as #f_num
while not eof(f_num)
    line input #f_num, li
    s = s & li & vbCrLf
wend
close #f_num
```

```
set f [open
set s [read
close $f
```

Read whole file into a variable. This VB code is very slow for even moderately large files. And it has no way to deal w
also normalizes different newline characters into a single kind of standardized newline character (by default). This cod
indicates 'read' mode.

```
dim a(1 to 3) as string
a(1) = "Mark"
a(2) = "Brian"
a(3) = "Roy"
'oops − need more elements
redim preserve a(1 to 10) as string
a(4) = "John"
```

```
array set a
set a(4) Jo
# now some
# element na
set a(Red)
set a(Linux
```

Array vs. Array. VB arrays are restricted to using numbers as subscripts (subscripts, or indexes, are called 'element nar
'ReDim Preserve' operation. Tcl arrays automatically expand, and they use a super−efficient hash table implementatior
data for an element name, and different styles can even be mixed within the same array. There are no restrictions on th
or through only certain elements in the array (by filter). You can also obtain a full or partial list of the element names,
in VB requires the use of a collection or dictionary object. Each of those comes with its own quirks and pitfalls, such a

```
(No equivalent)
```

```
array set my
set my_list
```

List to array, and back. Easy and rapid translation between these two primary data structures means that the tools for e

```
dim a(1 to 100) as string
dim i as integer
dim f_num as integer
f_num = freefile
open "my_file.txt" for output as #f_num
for i=1 to 100
    print #f_num, a(i)
next
close #f_num
```

```
set f [open
puts $f [ar
close $f
```

Write whole array. In this VB code, and frequently in other VB code, newlines and possibly other characters appearing
whenever your code deals with arbitrary data entered by the user. In Tcl they do not − the data is kept "clean" at all tim
(0x0A or decimal 10) characters are automatically normalized by default. Note that these two examples don't produce

be read back in (by Tcl) and automatically have the same number of elements, same element names, etc.. The Tcl list
non−ambiguous, textual representation. It is also readable and writable by humans.

```
(No equivalent)
```

```
set f [open
puts $f [ar
close $f
```

Write certain elements of an array. In the VB, a collection or dictionary object would have to be used for this. A loop
is `a` and a string pattern of `red*` (case sensitive) is used as a filter to select elements at high speed.

```
(No equivalent)
```

```
set my_list
```

Sort a list. The sort can be reversed, or ordered by numeric value, etc. It can also order a list of sublists using an index
lappend, **linsert**, **lreplace**, **lsearch**, **concat**, **split**, **join**, etc. Tcl lists can also be nested arbitrarily, and the **foreach** co

```
' requires a reference to ADO
' assume we have a connection called conn
dim rs as new recordset
rs.open "select id, name, age from people", _
    my_connection, adOpenStatic
' processing code goes here
rs.close
set rs=nothing
```

```
package req
# assume we
conn read a
# processin
unset a ;#
```

Retrieve a simple array of data from a database table. In VB data is always retrieved in a recordset object. In Tcl it can

```
(No equivalent)
```

```
package req
set httpTra
upvar #0 $h
if {$state(
        put
}
```

Retrieve a document or file from a web server.

```
(No equivalent)
```

```
regexp -all
```

Complex string pattern search and extraction. Tcl uses *regular expressions* for this. *Regular expression* is a specificati
'like' operator, except on steroids − a *whole lot* of steroids. Regular expressions are several times more powerful and fl
http://zez.org/article/articleprint/11. Tcl's regular expression parser is written in hand−optimized C code and is availab
powerful versions you're used to are also available for use in several different commands (**glob**, **string match**, **lsearch**
how tolerant of different situations it needs to be. In addition, that is some of the most difficult, error−prone, and slowe
the URLs of every image on an HTML page.

```
(No equivalent)
```

```
set find {<
set replace
regsub -all
puts $resul
```

Complex string pattern search and substitution. Again, Tcl uses regular expressions. This example would take 40 lines
maintenance programmer to follow it. And again, it is some of the most difficult, error−prone, and slowest code that c
systematically, while the contents of each cell is preserved.

```
(No equivalent)
```

```
set handle
set greeting
close $hand
```

Make a connection to a network socket (act as a client) and retrieve data. The example assumes a server is listening on

```
(No equivalent)
```

```
proc greetin
        puts
        clos
}
socket −serv
```

Implement a network server to answer the client shown above. This is the complete script. If you're using Wish (the Tc add a **vwait** command at the end, to make the program wait for events instead of terminating at the end of the script. Th default, and Tclsh is not.

3. Getting More Information

- *General Tcl/Tk programming and introduction:* See Brent Welch's unbelievable book *Practical Programming in Tcl and Tk*. Due to Brent's generosity, you can even read and print the older editions and selected chapters from the current editions at http://www.beedub.com/book .
- *Downloads needed to develop in Tcl:* See http://www.tcl.tk for TclPro 1.4.1 for all operating systems, plus almost any add−on package you could ever want. TclPro contains the 2 interpreters (Tclsh and Wish) version 8.3, plus an excellent interactive debugger and a suite of helpful tools and libraries. Version 1.4.1 was released to the public. However, as of mid−2002, it looks like ActiveState is taking over the TclPro product as a commercial product. Remember you can always get the 'standard' interpreters for all operating systems from http://tcl.sourceforge.net because Tcl is open source software.
- *Editors with syntax highlighting, etc:* For MS Windows, I like the inexpensive commercial product TextPad at http://www.textpad.com. Currently the cost is $27 US per license, and you can try before you buy. Be sure to get the Tcl syntax definition file from their web site. TextPad is the most feature−rich editor for MS Windows I've ever seen, and has the ability to emulate Microsoft editors' behavior. You can use it as an IDE for Tcl/Tk development by interfacing it with the interpreters and your other tools. For Unix/Linux, and maybe even for MS Windows, try Nedit at http://www.nedit.org. It's free under the GNU General Public License. It also does a good job of making MS Windows users productive right away.
- *Tools you'll probably want:* The first thing most VB programmers want is to hit an ODBC database. Go get the TclODBC 2.2 package from http://www.tcl.tk . It's a DLL for Win32 that hooks you into all ODBC data sources and drivers. It comes with documentation, and there's a minimal example above. Note that it may or may not be portable to other operating systems, so you might want to wrap all your calls to it into procedures. That way you can port your code to use other libraries later. Regular expressions are almost a powerful programming language of their own. Accordingly, they take some time to master. The simple Tcl program 'Visual RegExp' has helped me tremendously with that. Get it at http://laurent.riesterer.free.fr/regexp . There are also several packages available for hooking Tcl to the world of ActiveX, so you can automate MS Office applications, etc..
- *Essential help topics:* Once you have TclPro and its help file, go to its index and visit the 'Tcl' topic. There's a concise summary of the language's syntax rules, and the substitutions that drive it. Also be sure to hit the 're_syntax', 'tclvars', 'tclsh', and 'wish' topics. These are apparently translated from the Tcl man pages on Unix/Linux, and are some of the best texts I've ever seen for WinHelp, if you need *reference material*. I don't recommend reading this help file as your first introduction, but it is an excellent reference while programming.
- *'Start' menu items:* Once you have TclPro installed, you should look at the 'Start' menu for TclPro, and check out the 'Incr Widgets Reference' and 'Widget Tour'. These show the built−in GUI capabilities of Tk *with the actual Tcl code required to use them.*
- *Advocacy (how to convince your management to use Tcl/Tk):* A wealth of advocacy information is available at http://www.tcl.tk .

4. Copyright and License

"Visual Basic," "VBScript," and all related terms are trademarks of Microsoft − http://www.microsoft.com.

Tcl (Tool Command Language) is open source software, begun by John Ousterhout − http://www.tcl.tk or http://tcl.sourceforge.net.