



KEY CITY AMATEUR RADIO CLUB

April 2004

March Meeting Highlights

Kent West, KC5ENO

- Randy Robinson (N5JZH) called the meeting to order at 7:02pm.
- Susan Gerred of the City of Abilene then discussed the upcoming Steamin' Wheels Bike Race and Fun Ride, scheduled for March 20th; the routes have been changed this year due to road construction, such that the two longer routes are a couple of miles shorter each.
- Kent West (KC5ENO) then read the meeting minutes from the February meeting. Nesa Love (KC5ENL) motioned that the minutes be accepted as read, and Dee Burton (W5VRE) seconded.
- Peg Richard (KA4UPA) read the Treasurer's Report. She pointed out that the expense of printing this year's brochure was discounted by \$50, thanks to the printer, Jack Merck of MinuteMan Press. Jake Mullins (KC5GZP) motioned to accept Treasurer's Report as read; Jack Merck (KD5UVF) seconded.
- Peg then gave a report on Hamfest and on the latest VE Test Session.
- Bill Shaw (KJ5DX) gave a report on the Skywarn meeting in February, and said that on Thursday night, the 25th of March, there will be a meeting at 7pm at the View Fire Station for weather-net training. He also reported that the move of the Skywarn desk back into Fire Station #9 has been completed, except for phone and Ethernet connections.

- The club turned to discussing the upcoming Hamfest. Joe Marshall (WQ5M) and Peg Richard gave a brief history of the first Abilene Hamfest.
- A brief discussion was conducted concerning a beginner's ham class; there were eighteen people who signed up for such a class at the Skywarn training. The class will be scheduled after the signees have been contacted.
- Carla Dyer (K5RLA) had tendered her resignation as club president a few days before the March meeting. Randy tabled acceptance of her resignation pending discussions with her to reconsider. A letter asking her to reconsider was passed around and signed by attendees who concur with the letter's sentiment.
- Some discussion ensued about advertising the Hamfest and the KCARC in general by putting flyers around town, at area universities, making Kiwanis presentations, and having regular scheduled classes quarterly.
- Gary Peterson (NZ5V) motioned to adjourn the meeting; Charles Blythe KD5TKR seconded.
- Several club members met at McDonald's afterwards for coffee.

Club Meeting Reminder

Remember to attend the next meeting of the Key City Amateur Radio Club Monday, 12 April, at 7pm on the second floor of the Abilene Public Library.

Steamin' Wheels Bike Race

Kent West, KC5ENO

The twelfth annual Steamin' Wheels Bike Race and Fun Ride was conducted on Saturday, 20th March 2004, starting at 9am. Thanks to the following amateurs who provided communications or other assistance for the event:

Wayne Floyd, KK5XJ
Nesa Love, KC5ENO
Patricia Maloney, KD5PCY
Sherry Maloney, KD5FON
Tommy Maloney, KB5GAW
Randy Robinson, N5JZH
Kent West, KC5ENO
W.K. Wiggins, WB5ZOO
Jim Wilson, N7NGQ

Skywarn Meeting

Kent West, KC5ENO

Local ARES and RACES members and members of local volunteer fire departments met for a one-hour meeting on the evening of March 25th at the View Volunteer Fire Department. Emergency Coordinator Bill Shaw (KJ5DX) and Assistant EC John Bogart (WX5TX) instructed the attendees in the standard operating procedures used during weather net activations. 26 persons were in attendance. The Standard Operating Procedures manual is available in MS-Office, OO.o, StarOffice, and PDF formats from <http://www.qsl.net/kcArc/announce.html>.

Hamfest Next Month

Kent West, KC5ENO

It's time! The Key City Amateur Radio Club's 2004 19th Annual Hamfest will be on Saturday and Sunday, May 1-2, 2004 (first weekend in May). The hours on Saturday are from 8 a.m. to 5 p.m. (with setup at 6 a.m.), and on Sunday from 9 a.m. to 2 p.m. There will be three major prizes, and a Grand Prize Drawing on Sunday, along with hourly door prizes. Other features of this year's Hamfest:

- Full-service Concession Stand On-site
- Breakfast available Saturday morning at the KCARC Concession stand
- Drive-In loading & unloading
- Limited RV Hookups are \$ 5.00 per night
- Free basic test equipment bench and duplexer retuning
- Smoke-Free Environment
- Wheelchair access
- ARRL Volunteer Examiner Test Session
- QSL Card checking for WAS and VUCC for ARRL members (must contact the ARRL Awards Manager, John Dyer, AE5B (ae5b@ar-rl.net), in advance)
- Transmitter Hunts
- Saturday night fox hunt with special prize (special rules apply; get more information at the registration table)

Talk-In: 146.760 (PS 146.2)

Registration received by April 26th is \$8.00, otherwise it's \$9.00. Tables are \$ 8.00. Extra prize tickets are \$3.00. There is no Admission Charge, but you must register to be eligible for prizes. Registration also helps out the club.

For more information, such as a list of the major prizes or a PDF of the Hamfest flyer, or to download a registration form, visit <http://www.qsl.net/kcarc/hamfest.html>

Key City Amateur Radio Club Income and Expense Report March 8, 2004

<i>Regular Account:</i>	Balance Forward	\$504.10
	Income	90
	Expenses	68.37
	Balance on Hand	525.73
<i>Repeater Account</i>	Balance on Hand	64
<i>Hamfest Account:</i>	Balance Forward	1254.07
	Expense	278.61
	Balance on Hand	975.46
Amount Needed to Balance		4,565.19
Total Amount on Hand:		
	Checking	1529.06
	Cash	35
	Coins	1.13
	Total	\$1,565.19
** Itemized Expenses (Regular Account)		
	Checking Service Fee (Jan)	15
	Southwestern Bell (Mar)	22.19
	Postage March	5.18
	Annual Mailbox Rent	26
	Total	68.37

Peggy A. Richard, KA4UPA, Treasurer, Key City Amateur Radio Club

FCC Seeks Comments on Three Petitions

Kent West, KC5ENO

The FCC has received several petitions recently.

The first of these is RM-10867 and was filed by the ARRL. It proposes that there be only three license classes: a new "Novice" level that has limited HF privileges and would not require a Morse code test; the General class, into which current Technician, Tech Plus, and General licensees would be grouped, also not requiring a Morse code test; the Extra class, which would still require the 5WPM test.

The second petition is RM-10868 and was filed by the Radio Amateur Foundation. This petition also seeks to establish three license classes; the Technician class would have limited HF privileges; current Novices would be merged into the Technician class. The second class, General, and the

third class, Amateur Extra, would still be required to pass a 5wpm Morse code test. Current Advanced class hams would be upgraded to Extra.

The third petition is RM-10869 and was filed by Ronald D. Lowrance, K4SX. He's seeking to keep the 5 WPM Morse code requirement for the General class, whereas the Amateur Extra class would be required to pass a 13wpm Morse code test.

The fourth petition is RM-10870 and was filed by the National Conference of Volunteer Examiners. This petition seeks to create a new Communicator class which would essentially be a restoration of the Novice class, but without the Morse requirement. It would upgrade existing Tech and Tech Plus hams into the General class, and the Advanced class into the Amateur Extra class.

More info about these petitions is available at: <http://www.remote.ar-rl.org/news/stories/2004/03/24/2/?nc=1>

GUI Programming with Tcl/Tk

Kent West, KC5ENO

Introduction to Tcl/Tk

Who hasn't wanted to do a bit of programming on their Graphical User Interface (GUI, such as MS-Windows, Macintosh desktop, Linux X Window System)?

But the learning curve was always prohibitive.

Well now, using the Tcl/Tk tool, you can create cross-platform scripts that will run on any GUI platform that'll run Tcl/Tk.

Tcl stands for Tool Command Language, and is pronounced "tickle". It is a scripting language similar to shell scripting in Linux and Macintosh OS/X, and to the old DOS BASIC programming language. Tk (pronounced "tee-kay") is the Tool Kit that allows these Tcl scripts to easily create GUI applications.

Interpreter vs Compiler

Since Tcl is an interpreted language (like BASIC) rather than a compiled language (like C or Java), it runs slower, but except for speed-intensive applications such as games, that won't matter on modern computers.

When a program is interpreted, the computer reads an instruction in mostly-human-language, converts it into computer language, and then runs the converted instruction; then it reads the next instruction and repeats the process.

When a program is compiled, the computer translates the entire program at once and saves the result as an executable, such as an .EXE file; then the executable file is run.

Interpreted languages are easier to debug and prototype with, whereas compiled languages produce a faster finished product that is harder to modify later.

Each method has pros and cons; one of the pros of interpretation is that it is easier to move the program from a PC to a Mac to an IBM mainframe to a Palm Pilot, etc.

Installation

Before getting our feet wet with GUI programming using Tcl/Tk, it must first be installed.

Macintosh: Mac users will have to compile from source using Code Warrior, as per: <http://www.scriptics.com/software/tcltk/downloadnow84.tml>. It's good to know that Tcl/Tk is available on the Mac, but I'm just too lazy to try to compile from source (especially since I don't have Code Warrior), so I did not pursue the Mac any farther.

I wonder if it's available via fink on OS/X?

Windows: Go to:

<http://www.activestate.com/Products/ActiveTcl/> and click on the "DOWNLOAD" button in the upper right corner of the screen (below the ActiveSTATE logo).

You'll be asked to register, but I found that I could just bypass this by clicking on the `Next` button.

I grabbed the Windows version of the 8.5.0 Beta 1 release. It's about 9.4 MB, and once it was downloaded, double-clicked on the executable. I accepted all the defaults for an easy install.

If you don't want to install Tcl/Tk on your Windows computer, you can always use a Knoppix CD (<http://www.knoppix.org>) to temporarily boot into Linux to use this programming tool.

Linux: Probably in your distro. For example, Debian users can simply `apt-get install wish` to see which versions of Tk are available for your release, and then do something like `apt-get install tk8.4`.

You can also download a binary from the site mentioned above in the Windows instructions.

Getting Started

Create a text file (using Notepad on Windows, using your favorite text editor such as nano or vi on Linux). The text file should have the following text.

```
#!/usr/bin/wish
button .b -text "Hello World!" -command exit
pack .b
```

Save this file as "test.tcl" (make sure to put the name in quotes when saving in Notepad, to prevent Notepad from invisibly adding ".txt" onto the filename).

Windows users can now just double-click on "test.tcl" to run the program.

Linux users need to make the script executable with the command `chmod +x test.tcl`, and then can run the script with `./test.tcl`. If the script does not work, try running "which wish" and changing the first line in the script to reflect the path leading to the wish program.

You should see a small window pop up on your desktop with a button in it that says "Howdy, World!" If you click on the button, the window will exit.

Wow! Your first GUI application. Simple, wasn't it? I'll explain how this code works later.

Three Methods of Running Tcl/Tk

1 - The Tcl Shell

In Linux, at the command prompt, type `tclsh`, or in Windows, click on Start/Run, and enter `tclsh`.

This will open the Tcl shell, where you can enter commands directly. For example, enter the following command at the % prompt:

```
expr 2+4
```

You should see the result of 6 printed out. Here's another example:

```
% set name Kent
Kent
% puts "My name is $name"
My name is Kent
```

Well, that's all fine and dandy, but it's not GUI programming; that's just shell programming, and is boring.

Okay, to do GUI programming, we need to invoke the Tk half of the Tcl/Tk toolset. First, exit out of tclsh with:

```
%exit
```

2 - The Tk shell, aka wish

Now fire up the Tk portion, called "wish" by starting wish (at the Linux prompt, or at the Start/Run prompt in Windows).

This will result in a terminal window with a % prompt, like you saw with tclsh, and a graphical window (or "widget") named "wish" on your desktop.

If you try the above tclsh examples, you'll see the same behavior, and the "wish" widget remains unchanged.

But if you use the commands we used in our test.tcl script farther above, you'll see some activity.

```
%button .b -text "Howdy, World!" \
    -command exit
%pack .b
```

(Notice you don't include the `#!/usr/bin/wish` line. By the way, the symbol `#!` is usually referred to as a "shebang", or sometimes as a "splat-bang".)

Notice that the first command (the one that starts with `button .b`) was too long to fit on one line, so I used the continuation character of `\` which allows the two lines to be treated as one.

The main widget window (titled "wish") shrinks down to just fit the newly created button widget (labeled "Howdy, World!"), and when you click on the button, the entire widget closes and the wish terminal closes.

3 - wish Scripts

Although running Tcl commands from the wish shell makes it easy to prototype and test various commands, it's not so convenient for doing repetitive tasks, which is what scripting is all about.

So rather than running individual commands from the wish shell, we can write scripts, and then just run the script. This is what we did early on with our "test.tcl" script. In a few minutes we'll create a more complicated script that converts temperatures from Fahrenheit to Celsius and vice-versa.

The Explanation

But first, a brief explanation of the script we wrote earlier.

The first line, `#!/usr/bin/wish` tells the operating system to run the program `"/usr/bin/wish"` to interpret the rest

of the script. This line doesn't do anything on Windows computers; instead, on Windows computers the script needs to have the `.tcl` extension on the filename. Linux computers on the other hand, need this line, but the filename extension can be anything (or nothing).

The next line, `button .b -text "Howdy, World!" -command exit`, defines an object (tcl is *almost* object oriented, for you programmer types, but not quite). This particular object is of a certain kind, or "class", called a button. There are other types (or classes) of object also, such as labels, and frames, etc. This button object has a name, ".b". You could have called it ".myButton" if you wanted to. Object names must start with a dot followed by a number or a lower-case letter. The dot also acts as a separator in a "path" of objects, much like the `/` in Linux or the `\` in Windows acts as a separator in a path to a file. Since the window widget which first appears on the GUI screen when you start wish is the "root" widget, its name is simply ".".

In addition to having the name ".b", this button has been instructed to print some text on the button containing the words "Howdy, World!". And when the button is pushed, it will execute a command that exits the program.

Edit your script, and try this modification:

```
#!/usr/bin/wish
button .b -text "Howdy, World!" \
    -command exit \
    -background black -foreground yellow
pack .b
```

Notice that the first command line got too long to fit in my editor without wrapping, so I added the `\` on the first two lines of the first command to indicate that these three lines should be treated as one

Now run the script, and you'll see that you have different colors than before.

The third line, `pack .b`, actually causes the button named ".b" to be displayed on the screen. The second line only creates the button; it does not cause the button to be displayed. So after the second line, the button exists in memory, but not on the screen. It's not until the button is packed that it exists on the screen.

That should give you a half-decent feel for what's going on. Now we're ready to move onto a real application.

The Temperature Conversion App

Create a new script, named "c2f2c.tcl" (or whatever you want to call it) with the contents as shown in Listing 1.

We've added quite a bit here. First, notice that you can put comments into your scripts with the splat character. I highly recommend you comment your code; it'll not only help other people to understand your program, it'll help you to understand it when you come back to it several months from now.

```
#!/usr/bin/wish

#
# This program converts temperatures from F to C and vice-versa
# [Your Name]
# {Today's Date}
#

#
# Create the Fahrenheit entry box, label, and display them
#
entry .fbox -width 6 -textvariable ftemp -relief sunken
label .flabel -text "Fahrenheit"
pack .fbox .flabel -side right -padx 5 -pady 5
```

Listing 1

The `entry` line defines a new object. This is an entry box, where the user can enter information. Its name is `.fbox`, but it could be `.boxForTypingInFahrenheitTemperature` if you wanted it to be. This entry box will have a width of 6 characters (roughly), and whatever gets typed into the box will go into a variable named "ftemp". Finally, the box will have a sunken look to it. Feel free to play around with these options to see what happens.

A second object of class `label` is created, named `.flabel`. Unlike a button that can be pressed, or an entry box that can have information typed into it, a label simply displays text, in this case "Fahrenheit".

Finally the box and label are displayed with the `pack` command. You could have two separate `pack` commands, like so:

```
pack .fbox
pack .flabel
```

but we combined them onto one line for convenience. We also specified that the `.fbox` would be shifted to the right edge of the root widget, followed by the `.flabel` which would be shifted to the right edge without bumping into the previous object. We've also added some extra spacing around the objects to make the resulting window look a little nicer.

Now we need to do the same thing for the Celsius box; just add the code that's in Listing 2.

```
#
# Create the Celsius entry box, label, and
# display them
#
entry .cbox -width 6 -textvariable ctemp -relief sunken
label .clabel -text "Celcius"
pack .cbox .clabel -side right -padx 5 -pady 5
```

Listing 2

Now run your script, and you should see both entry boxes.

It might be nice to have an Exit button, so go ahead and add the code in Listing 3.

```
#
# Create the Exit button and display it
#
button .exitButton -text "EXIT" -command exit
pack .exitButton -side right -padx 5 -pady 5
```

Listing 3

Notice that the exit button is on the left side of the root widget; it's as far to the right as it can go without bumping into the previous objects. To get it to the right, we need to display it first, so I've rearranged the code as in Listing 4.

```
#!/usr/bin/wish

#
# This program converts temperatures from F to C and vice-versa
# [Your Name]
# {Today's Date}
#

#
# Create the Exit button and display it
#
button .exitButton -text "EXIT" -command exit
pack .exitButton -side right -padx 5 -pady 5

#
# Create the Fahrenheit entry box, label, and display them
#
entry .fbox -width 6 -textvariable ftemp -relief sunken
label .flabel -text "Fahrenheit"
pack .fbox .flabel -side right -padx 5 -pady 5

#
# Create the Celsius entry box, label, and display them
#
entry .cbox -width 6 -textvariable ctemp -relief sunken
label .clabel -text "Celcius"
pack .cbox .clabel -side right -padx 5 -pady 5
```

Listing 4

Now when you run this program you should see a window like so:



Now for the actual procedures that convert the temperature.

We need to create a new command for the conversion each direction. The new command for going from F to C is in Listing 5.

```
#
# f2c - This converts a temperature from Fahrenheit to Celsius
#
proc f2c {ftemp} {
    set result [expr ($ftemp-32) * 5/9 ]
}
```

Listing 5

The new command is a "procedure", and its name is `f2c`.

The `{ftemp}` tells the procedure that whenever this procedure is invoked by some other part of the program, it will also be given some value, which will be put into the variable `ftemp`. This value is the Fahrenheit temperature to be converted to Celsius, and is referenced in the formula as `$ftemp`.

When variables are set, they are set using just the name; when they are read, they are referenced by the name with a dollar sign in front of the name. This short snippet of code (which is just for explanation purposes, and should not be in your script) should make this a bit more clear:

```
set myName "Kent West"
puts $myName
```

will print out

```
Kent West
```

The `f2c` procedure could consist of several lines grouped together by the curly braces, but in our case, it's just a single line within the curly braces. This single line assigns a value (using the `set` command) to the variable `result`, which is composed of the mathematical expression $(F-32) * 5/9$ which is the standard formula for converting from F to C. When the procedure finishes, it returns whatever value is in the `result` variable.

At this point in our program, the procedure is just loose code that has no function; no other part of the program is making use of the procedure. We've defined it, like we defined our button earlier, but haven't yet "called" it into action, just like we don't call our button into action until we display it with the `pack` command.

In order to call our procedure, we need some sort of event to activate it. That event could be a mouse click on a button that says "Calculate", but to keep things simple, we'll just cause the ENTER button on the keyboard to be the activating event.

In order to do that, we need to "bind" the ENTER (or <Return>) key to the Fahrenheit entry box, and use that binding to call the new procedure. This is done with the following code:

```
bind .fbox <Return> \
    {set ctemp [f2c $ftemp]}
```

This means that whenever the focus is on the Fahrenheit box (named ".fbox") and the ENTER key is pressed, the instruction in curly braces will be executed.

One of the hardest things about tcl/Tk is learning when to use curly braces, when to use brackets, when to use parentheses, and when to use quotes. I won't try to explain that now.

The command in the curly braces will set the variable `ctemp` to some

value. Since the variable `ctemp` is associated with the `.ctemp` entry box, whenever we set this variable, the ".ctemp" entry box will display this value. Entry boxes are interesting in this manner; they are a two-way street, in that you can type a value into the

entry box, thereby setting the value of the associated variable, or you can set the associated variable with a value, and that value will be displayed in the entry box. Works to our advantage here.

The value that is given to `ctemp` is

```
#!/usr/bin/wish

#
# This program converts temperatures from F to C and vice-versa
# [Your Name]
# {Today's Date}
#

#
# f2c - This converts a temperature from Fahrenheit to Celcius
#
proc f2c {ftemp} {
    set result [expr ($ftemp-32) * 5/9 ]
}

#
# c2f - This converts a temperature from Celcius to Fahrenheit
#
proc c2f {ctemp} {
    set result [expr $ctemp * 9/5 + 32 ]
}

#
# Create the Exit button and display it
#
button .exitButton -text "EXIT" -command exit
pack .exitButton -side right -padx 5 -pady 5

#
# Create the Fahrenheit entry box, label, and display them
#
entry .fbox -width 6 -textvariable ftemp -relief sunken
label .flabel -text "Fahrenheit"
pack .fbox .flabel -side right -padx 5 -pady 5

#
# Create the Celcius entry box, label, and display them
#
entry .cbox -width 6 -textvariable ctemp -relief sunken
label .clabel -text "Celcius"
pack .cbox .clabel -side right -padx 5 -pady 5

#
# Bind the entry boxes to specific actions when ENTER is pressed
#
bind .fbox <Return> {set ctemp [f2c $ftemp]}
bind .cbox <Return> {set ftemp [c2f $ctemp]}
```

Listing 6

determined by what's in the brackets. In this case, that's the call to our newly created procedure `f2c`. We call the procedure and give it the temperature to convert. If the temperature to be converted is 45, the command would look essentially like:

```
set ctemp [f2c 45]
```

which would be the command executed when the <Return> key is pressed while the cursor is in the “.fbox” entry box.

At this point, your program should convert from Fahrenheit to Celsius, but not the other way around. In order to do that, we need to add another procedure and another binding for the C to F conversion.

Also, as a general rule, you want your procedures at the top of the script, so I'll again rearrange the script a bit, and I'll add in the C to F parts, so that the final script is in Listing 6.

You can now convert temperatures all day long. And you can move the

script from one platform (ie Windows) to another (ie Linux) without any "porting" being involved. You can tinker with fonts and colors, and expand the program to convert to Kelvin, or whatever. Have fun!

Tidbits

Kent West, KC5ENO

* In a letter to club members in March, KCARC President Carla Dyer (K5RLA) resigned from her position. Many club members expressed regret at her resignation, but understanding of her need to do so.

* The city of Abilene has requested that the KCARC provide communications for the upcoming Formula One Duathlon, to be held at 9am on Sunday, 24th October 2004. The participating professional marathoners will be running, then biking, then running, then biking, and then running once more. It will be conducted in the area of the Expo Center, and is sponsored by some big names like Gatorade.

* As mentioned by David, KC5PPI, on a recent Thursday night's weekly Information Net on the 146.76 repeater, local television station KRBC now offers a service whereby you can be alerted to severe weather via email. If you have an email-enabled text messaging device, such as a cell phone, text pager, PDA, etc, you can get instant alerts using this free E-Warn service provided by KRBC. You can get more info and sign-up at KRBC's website,

<http://www.krbc.tv/ewarn.htm>.

* This past December, three young-earth creationists presented posters at the American Geophysical Union Fall Conference in San Francisco, attended by about 10,000 scientists from all fields of geophysics. According to <http://www.icr.org/research/misc/agu-conference.html>, the three papers, presenting “solid evidence for a completely different paradigm about the age of the earth”, “were controversial but . . . well received by those who interacted with the authors.”

Key City Amateur Radio Club, Inc.

P.O. Box 2722

Abilene, TX 79604

First-Class
Postage Required

Forwarding Requested

First Class Mail

Newsletter Staff

Editor:

Kent West, KC5ENO

Circulation:

Peg Richard, KA4UPA

Jim Richard, K1UQI

Contributing Editors:

Whoever . . .