Internal Structure of the MorseKOB Program

Les Kerr

6 June 2006
Revised, 2 June 2008

The MorseKOB program contains three modules: KOB, Translator, and Internet. These modules run as separate processes, each in its own thread. In addition, a Windows form handles most of the user interface functions.

The KOB module performs the following functions:

- Checks the status of an external key (bug, straight key, or paddle)
- Provides a keyer function to simulate a bug
- Drives an external sounder
- Sends simulated sounder waveforms to the speakers
- Ties together the inputs and outputs of the other modules

The KOB module is the heart of the program, and it's actually the simplest of the modules. Its thread is assigned a higher priority than the other threads to ensure the best possible performance with the key and sounder. The module consists of a continuous loop, executed every 4 ms. On each cycle of the loop, the module checks the input lines of the serial port and determines the correct sounder state (mark or space) depending on the state of the key, the KOB circuit closer, the code sender, and the internet loop.

Whenever the sounder state changes, the KOB module drives the serial port accordingly and sends the appropriate click or clack waveform to the speakers. It also sends the sounder state to the code reader and to the internet module.

Keyer

The only code of any complexity in the KOB module is that which allows an external paddle to simulate a bug. This turns out to be trickier than it sounds; it's more than just sending a string of dots when the dot contact is closed and closing the circuit when the dash contact is closed. For example, it includes a feature that forces at least one dot to be sent whenever the dot contact has been closed, even when the dot contact is already open by the time the dot should be sent. This is an attempt to make the keyer paddle behave in a manner that's familiar to an operator accustomed to using a bug.

Code sender

The translator module consists of three components: the code definition table, the code sender, and the code reader (decoder).

The philosophy behind the code sender is to generate Morse with a rhythm that's similar to what's heard on the wire, but it doesn't go so far as to try to fool you into thinking there's a human operator at the key. It's mechanical and predictable, but also fairly natural.

The basic time unit is the length of a dot, in milliseconds, which is determined by the Code Speed setting according to the formula

$$DotLen = 1200 / WPM$$

The lengths of the various code elements, in terms of dot lengths, are shown in Figure 1. Relative to the 1.0 units of the dot and the element space, the lengths of the other elements are longer than you might expect. Another way to look at this is that the dots have been "sped up" relative to everything else, a practice that is frowned on by many International Morse operators but very common among bug users.

Code Element	Relative Length	Code Element	Relative Length
Dot	1.0	Element space	1.0
Dash	3.3	"Morse" space	2.7
L	5.6	Character space	3.7
Ø	8.9	Word space	7.0

Figure 1. Code element lengths

In some situations, adjustments are made to the lengths given in Figure 1. Certain characters – the letter L, the numerals, and the punctuation marks – are "emphasized" by adding extra space before and after the character: 4.2 dot units instead of 3.7. Similarly, in a string of dashes, the space between the dashes is lengthened from 1.0 to 1.6 units, while the length of each dash in the string is shortened from 3.3 to 2.7. This reflects the fact that back-to-back dashes, which have to be made manually on a bug, aren't normally spaced as closely as dot strings or dot-dash and dash-dot sequences.

Code reader

Other than the decoder algorithm itself, which is described in the MorseKOB technical paper *An algorithm for decoding American Morse*, the main challenge for the code reader has been to keep it from interfering with the bug and the sounder. For some reason, Visual Basic can take many tens of milliseconds to display a character in the code reader window, which is enough to disrupt the critical timing of the rest of the program.

I don't know why the display update should take so long, and I don't know of any other way to add a new character to the display, so I've had to look for other ways to get around the problem. The first solution was to wait for the gap between words before updating the display, instead of displaying a character at a time. Now the approach is to run the code reader in a separate, low-priority thread, so it won't block the time-critical parts of the program.

Internet module

The KOB program uses the same method for sending Morse over the internet as John Samin's CWCom program. John's scheme is fairly straightforward, but also very ingenious. Each string of dots and dashes that makes up a Morse character is sent in its own internet packet, with marks represented by positive numbers (for the length of the mark, in milliseconds) and spaces by negative numbers.[1] The server then rebroadcasts the packet to all other stations currently connected to the same Morse wire as the sender. Upon receiving a packet from the server, the KOB program reconstructs the Morse character exactly according to its original timing.

CWCom (and therefore also MorseKOB) uses the UDP protocol for sending packets over the internet. Unlike the more robust TCP protocol, it's possible for UDP packets to get dropped in the internet before they reach their destination. To guard against this, the Internet module sends a duplicate copy of each packet and incorporates a sequence number into the packet so the receiving program can ignore the duplicates. Despite this precaution, when the internet is congested it's still possible for the original packet and its duplicate to get lost. When the KOB program detects a gap in the sequence numbers of the received packets, it alerts the operator by displaying an asterisk in the code reader window.

MorseKOB includes some extensions to the CWCom communications protocol in order to support the closed-circuit style of landline telegraphy. These extensions are described in the MorseKOB technical paper *Extending the CWCom communications protocol to support closed-circuit telegraphy*.

Operating modes

The KOB program provides four possible operating modes: *normal*, *keyer*, *loop*, and *modem*. Loop mode is depicted in Figure 2. As its name suggests, loop mode corresponds closely to the traditional loop architecture of the Morse telegraph.

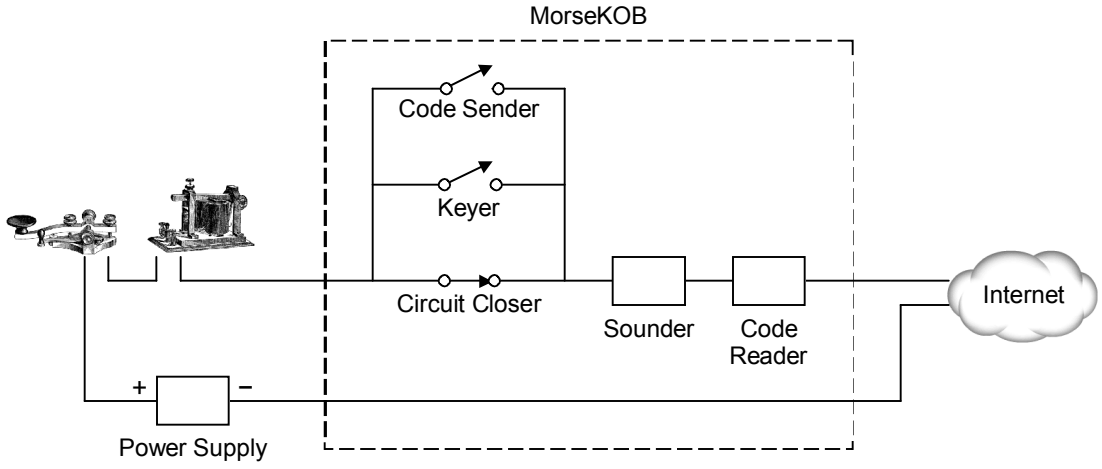

Figure 2. MorseKOB in "loop" mode

[1] Internally spaced letters (C, O, R, Y, and Z) may actually be sent as two packets instead of one, since the Internet module doesn't try to distinguish between "Morse" spaces and character spaces.

There are several differences between normal mode and loop mode: (1) In normal mode, the external key is treated as though it's connected in parallel with the KOB code sender, keyer, and circuit closer, and the external sounder is effectively in series with the KOB internal (simulated) sounder. (2) In normal mode, the sounder output on pin 7 of the serial port follows the key input on pin 6, whereas in loop mode the sounder output is isolated from the key input.[2] (3) Normal mode uses a contact debouncing algorithm for input conditioning, while loop mode uses a noise filter algorithm instead.

Keyer mode works the same as normal mode, except the program assumes keyer paddles are connected to the serial port and these inputs are used to control the KOB keyer function.

Modem mode is identical to loop mode, except the program inverts the polarity of the input and output signals of the serial port for compatibility with the standard for modems.

[2] The external sounder output is isolated from the external key input in loop mode (and modem mode) for compatibility with loop interface circuits and dialup hubs.