CHANNEL SURFING WITH NEWTON

Paul R. Potts
potts@pharos-tech.com

In February Apple began distributing a preliminary draft of a new toolkit and API which allows Newton applications to control the Newton's infrared LED. Developers have heard that the Newton is capable of broadcasting infrared radiation at the same frequency used by many consumer-electronics devices, but until now, without a method to control the infrared hardware, there was no way to take advantage of it. By the time this article appears in print, these files should be available to all Newton developers. *[With the introduction of NTK 1.0.1 the IR Toolkit functions are built into the Platforms file. The sample files referred to by Paul can be found on Apple's ftp site and on AppleLink. Ed.]*

Even with preliminary code, PIE Engineering has come up with a very easy-to-use demonstration application. The sample controls Sony and Philips CD players, but includes additional resources for Sony TVs, so I was able to use the resources given and figure out how to modify the basic code to add support for my Sony television in an hour or two. The Newton can let me channel surf at quite some distance: over twenty feet (the limit of my apartment). The preliminary code did have a tendency to hang the Newton after a few dozen commands, but I expect the occasional crash in preliminary code.

GETTING STARTED

You must replace your current MessagePad platform file (for NTK beta 7) with the new one included in the Remote Toolkit — it contains the IR transmission methods. There are just three commands, making this an extremely simple API:

- `:OpenRemoteControl()` returns a "magic cookie" that you send to the other functions;
- `:CloseRemoteControl(cookie)` returns `nil` and invalidates the `cookie`; and
- `:SendRemoteControlCode(cookie, command, count)` repeats the given command for count times.

All three of these are actually method calls and not global functions, so they must be preceded with a colon.

The basic sample project comes with four files installed: the layout file containing the code, and three resource files. The code sample shows a clever method for referring to resources. A frame, `irCodes`, contains three arrays of symbols which reference resources read by NTK and assigned to constants defined in the program's Project Data file.

IR COMMAND FORMATS

The source code for the resource files, in Rez format, is included, and is nicely commented. The file "RemoteTypes.r" contains a template for 'IRCD' resources, which define the encoding for a particular command. Let's take a quick look at the "Play" command for a Sony CD player. (This information is taken from the comments in the "Sony.r" file by Mike Cremer, but I think it might be more easily understood if drawn with non-ASCII graphics).

The underlying timebase of all Sony IR commands, one count, is 600 microseconds. A *mark* code is one count idle plus four counts on. A zero bit consists of one count low plus one count high; a one bit consists of one count low plus two counts high.

The Sony "play" command has a seven-bit command field plus a five-bit device field (so that your system can tell the difference between "play" for your casette deck and "play" for your CD player). The command for "play" is 0110010; the device code for a CD player is 10001. A complete IR command stream consists of a mark signal, followed by the concatenated device and command fields (100010110010) sent backwards, low bit to high bit. See Figure 1 one for a visual representation of the play command.

This command is represented by the following resource definition in the "Sony.r" file from the sample code:

```
resource 'IRCD' (200, "SonyCDPlay") { 'cdpl',
600,  // timeBase in microseconds
4,    // leadIn, in timeBase units
74,   // repeat, in timeBase units (~ 44 ms)
833,  // leadOut, in timeBase units (~ 500 ms)
  {
    1, 1, 1, 2,
    1, 1, 1, 1,
    1, 2, 1, 2,
    1, 1, 1, 2,
    1, 1, 1, 1,
    1, 1, 1, 2
  }
}
```

The `repeat` value is only sent between repeating commands; `leadOut` is only sent after a non-repeated command ("play" is not a repeating command).

The data itself is represented rather oddly in the resource. I think of it like this: each number represents a state change, and the LED then remains in that state for the specified number of `timeBase` units. Thus, the first zero bit is represented here by "1, 1". (The first one represents a transition from the previous state, "mark," which is on, to off, and a wait of one time unit; the second one represents a transition back to "on" and one time unit on). The first one bit is represented by "1, 2" (transition from the previous state, one time unit wait, transition, two time unit wait).

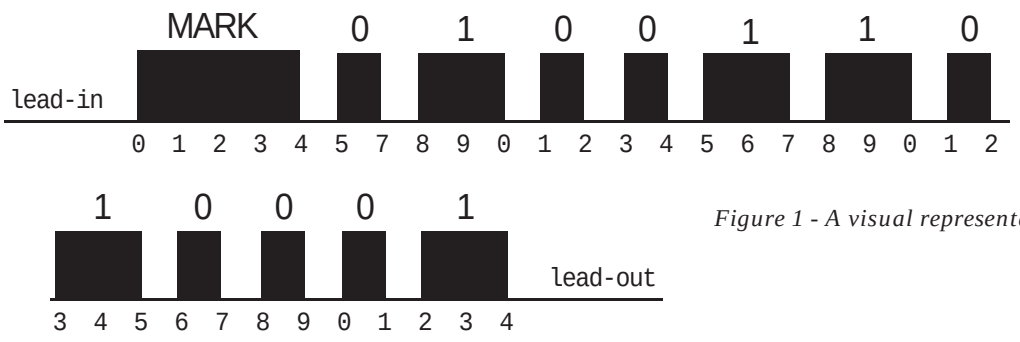

Figure 1 - A visual representation of the play command.

This representation scheme may look complicated, but together with the `timeBase` allows codes from any vendor to be represented. For a more challenging example, take a look at the Philips (RC-5) codes in the sample files. These codes are represented with the same resource scheme, but use a much different encoding. As an added twist, when a Philips command is repeated, its bits are inverted each time. This requires two resources for each command – one for each state, and a means of toggling between the two. The sample application demonstrates a way to do this.

IR FUTURES

How programmers using the Newton Toolkit for Windows will work with such resource-based data remains to be seen; maybe Apple will port the standalone Rez tool to Windows. This wouldn't allow editing of resources like PICTs, though.

PIE has said that they aren't planning to provide codes for every vendor's devices. Instead, they have tried to give developers a chance to share information by setting up an Internet mailing list for IR hackers, NewtonIR@blammo.apple.com. To get on the list, send a message to NewtonIRrequest@blammo.-apple.com. Discussion on this list was quite heavy for a while but has tapered off recently. Support on AppleLink is also available in the PIE Developer Talk board; DTS suggests you include "IR" in the message subject.

You may notice that it should be possible to come up with a simple function to generate these codes at run-time, perhaps a function for each vendor's encoding type, instead of storing the resource data in the project. This seems feasible as long as the function can spit out the data fast enough to send the codes at the correct rate (including repeating codes; for example, holding down the volume button should send a continuous stream of volume-down commands). This topic is currently under discussion on the NewtonIR mailing list.

One thing to keep in mind: before you get too excited about turning the Newton into a universal programmable remote control, please note that while the MessagePad can beam infrared radiation at the proper frequencies, it can't receive it. According to PIE DTS, a filter designed to improve beaming using the Sharp protocol only allows a narrow band of IR to be received. ▲

Paul R. Potts lives in the frozen wastes north of Cincinnati. When not waving his Newton in the faces of complete strangers and babbling excitedly about the digital convergence, he can be found in his slightly dented Plymouth Reliant, stalled at the end of the on-ramp leading to the Information Superhighway.

Note: The IR protocol used for performing remote control functions with a Newton is not the same protocol that is used for beaming serial data through a standard Newton communications endpoint. This protocol is defined by Sharp. Contact Robert Stuart, Sharp Electronics Corporation (stuart@secms.-sharpwa.com) for details.

It should be possible to come up with a simple function to generate these codes at run-time, perhaps a function for each vendor's encoding type, instead of storing the resource data in the project.