Livewire Control Protocol on the Z/IP ONE

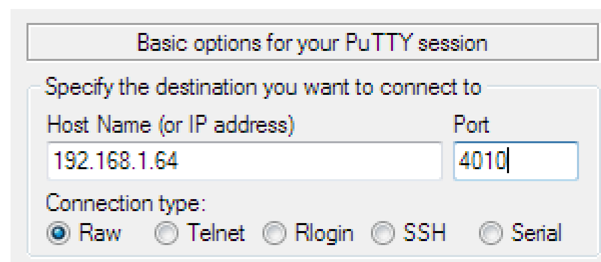
10 August 2017, Cleveland Ohio, USA

Users of Telos' earlier codec products may remember using the telnet command line interface to automate control of the unit. While this worked well for many purposes, the telnet interface was always intended for interactive control, and was not always maintained as an application programming interface.

Remote control on the Z/IP ONE is primarily accomplished using Livewire Control Protocol (LWCP). This is a protocol shared by many Livewire-enabled products, though there are objects unique to the Z/IP ONE.

LWCP basics

Livewire control protocol is an entirely text-based protocol operating on a raw socket, TCP 4010. Although it is intended for use as a scripted, automated control interface, it can be used interactively for development, testing, or for simple tasks. Here are the settings that you would use if connecting via the popular terminal program PuTTY:



LWCP is a fully defined protocol, and the latest reference document is attached below. The latest Z/IP ONE implementation document is also attached.

LWCP Reference v.13

LWCP for the Z/IP ONE v1.5

At its most basic, you have a set of verbs, which work on options, given parameters. For example get supv state means that you want to get the state parameters from the supervisor object. Similarly, login unit user="user",pwd="" sends a login request to the "unit" object with a username of "user" and no password.

Examples

These are technical documents, but the protocol looks relatively simple in practice. Here is a very short sample sequence:

```
get supv state
indi supv state="WORKING",version="4.0.1b",device_type="ZIPOne",hwversion=2 $status=OK
login unit user="user",pwd="" $ack
ack unit $status=OK
get phonebook buddylist
indi phonebook buddylist=%BeginEncap%<list><buddy><name>ZephyrIP10</name>
<group>public</group>
<pwd>public</pwd>
<nick>Telos Line</nick>
<type>tscp</type>
<panic>0</panic>
<status>online</status>
<redial>global</redial>
</buddy>
<buddy><name>euroline</name>
<group>public</group>
<pwd>public</pwd>
<nick>Euro Line</nick>
<type>tscp</type>
<panic>0</panic>
<status>online</status>
<redial>global</redial>
</buddy>
</list>
%EndEncap% $status=OK
```

In the above listing, lines in blue are sent by the operator or control program. The lines in orange are responses from the Z/IP ONE. As you can see, the “supv” object can be accessed before logging in. In this way, your control program can connect to a LWCP-enabled device and determine the type and version of the product in case special control is needed.

Any command can be given special parameters `$ack` and/or `$trxi=”asdf”`. Respectively, these request an acknowledgment, and specify a transaction ID. The acknowledgement can be used to verify command receipt when a command would not usually generate a response. Similarly, the transaction ID can be any user-specified string, and can be used to establish which responses correspond to which commands. Again, an example may clarify:

```
set network wanip="172.16.237.69"
set network wanip="172.16.237.69" $ack
ack network $status=OK
set network wanip="172.16.237.69" $trxi="as df"
set network wanip="172.16.237.69" $ack $trxi="as df"
ack network $status=OK $trxi="as df"
set network wanip="172.16.237.69" $ack $trxi=1234
ack network $status=OK $trxi=1234
```

Most `set` actions do not generate a response by default. The first command was successful. If your application needs to know whether the command succeeded or not, it can request acknowledgement. In this case, the `$status=OK` indicates that the command completed successfully.

Next, you can see how transaction identifiers work. The first line that uses a transaction ID ($trxi) still does not receive a response. That is because the command does not generate responses by default (if it had, perhaps due to an error condition, the transaction ID would be included). On the next try, by forcing a response with the $ack parameter, the transaction ID is returned. This identifier is arbitrary, and is intended to be used by the controlling application to track a “conversation.” As shown by the last pair of lines, whatever you specify as the $trxi parameter is echoed back on the response. You can use this to write applications that work asynchronously to the communication channel, yet can still detect responses to specific messages, reordering events, etc.