INTERFACING DISPARATE SYSTEMS

DON'T LET THE LACK OF A KNOWN INTEGRATION METHOD KEEP YOU FROM ACHIEVING YOUR GOALS. IF YOU HAVE TO, BUILD IT YOURSELF!

JAMES LITTON

When hearing the word interface, most people probably think of a Graphical User Interface or a physical hardware interface (serial, USB). If you dabble in scripting or are a serious developer, you, no doubt, are familiar with the concept of software interfaces as well. Occasionally, the need arises to integrate disparate systems where an interface doesn't already exist, but with a little ingenuity, an interface can be created to bridge the disparity and help you meet your specific needs.

I have an extensive home automation implementation I developed over the years. As I knocked out the "easy" integrations, I eventually came to a point of wanting to integrate systems that are not home automation-friendly. An example of this is my alarm system. Excellent alarm panels exist on the market that make integration a cinch, but I already had a fully functional alarm system and was determined to integrate it into my home automation setup rather than replace it.

My first inclination was to hack a keypad or build my own hardware interface that would allow me to capture status information. Both of those approaches are viable, but as I thought about other options, I realized I could integrate my proprietary alarm system into my home automation system without even cracking open the alarm panel.

Before I reveal the details of how I achieved the outcome I wanted, let me first lay out my integration goals. Although it would be nice to capture sensor data from the alarm system, in my case, it was totally unnecessary as the only data that might be helpful was motion sensor data or specific zone faults. Because I already have numerous motion sensors installed that are native to my home automation build, and because fault data wasn't a factor in my immediate integration requirements, I concluded that I needed to know only if my alarm was "armed" or "unarmed". Knowing the state of the alarm system helps me make my home automation system smarter. An example of this added intelligence might be to change the thermostat setting and turn off all lights if the alarm state changes to armed. Another example might be to turn on all of the lights in the house when the garage door opens after dark and the alarm is armed.

EXCELLENT ALARM PANELS EXIST ON THE MARKET THAT MAKE INTEGRATION A CINCH, BUT I ALREADY HAD A FULLY FUNCTIONAL ALARM SYSTEM AND WAS DETERMINED TO INTEGRATE IT INTO MY HOME AUTOMATION SETUP RATHER THAN REPLACE IT.

As I thought through the scenarios a bit further, I quickly realized I needed a bit more data. Depending on how an alarm system is installed and the practices of its users, there may or may not be multiple armed states that need to be considered. In my case, I have two separate armed states. One state is "armed away" (nobody home) and the other is "armed stay" (people are in the house). It wouldn't make sense to turn off all of the lights in the house, for example, if the system was set to armed stay, but that would make perfect sense if it were set to armed away. As I continued to think through my needs, I concluded that knowing whether the system was armed away, armed stay or unarmed was all I needed to add significantly greater intelligence to my home automation scenes.

Once I had a firm grasp of my needs, I realized my alarm-monitoring company already was providing me with some of the data I was looking for in the form of e-mail messages. Every time the alarm was armed or disarmed, I would get an e-mail message indicating the state change. I had been using this feature for a while, as it was helpful to know when my kids arrived home or when they left for school in the morning. Because I had been using this notification mechanism for some time, I also knew it to be extremely timely and reliable.

Because I was getting most of the data I needed, I started thinking about ways I might be able to leverage my e-mail system as the basis for an interface to my proprietary alarm panel. In days gone by I had used procmail to process incoming e-mail, so I knew it would be fairly easy to inject a script into the inbound mail-processing process to scan content and take action.

Before I started down the path of writing a script and figuring out how to make my e-mail system run inbound mail through it, I needed to deal with the shortcoming I had with status notifications. You may have noticed that I said my alarm monitoring company was sending me two status notifications: one for armed and one for unarmed. I was fairly certain that an additional relay could be configured so the folks at the company could notify me with the two variations of "armed" that I needed to proceed, so I called them to discuss the matter, and sure enough, they were able to make the change I requested. In fairly short order, I was receiving the three notifications that I wanted.

With the notifications in place, I could start the task of creating a script to scan incoming mail.

To keep things as simple as possible, I decided to write the script in Bash.

To follow this example, the first thing you need to do is capture all of the data being piped into the script and save it for processing:

```
#!/bin/bash
while read a
do
  echo "$a" >>~/tmp/results.tmp
done
```

This block of code redirects inbound e-mail messages to a temporary file that you now can perform search operations against. Because e-mail messages are simple text files, there are ample methods you can leverage to search for the strings that tell you the state of the alarm system. In this case, you could receive three possible messages which are "Your alarm has been Armed Stay", "Your alarm has been Armed Away" or "Your alarm has been Disarmed".

Now that you know exactly what you are looking for, use grep to perform the search operations. When you combine your grep searches with if statements, you can execute a specific command when one of your searches evaluates to true.

The heart of my home automation system is a software package that has an extensive REST API that I can leverage to change device states, set variables, control access groups and control device links. This makes it extremely easy to set a variable for the alarm state that I then can use to trigger various actions and control scenes in my home. To interact with the REST API, let's use curl.

In my case, my home automation software expects data to be sent as a PUT instead of curl's default of GET. To accomplish this, let's use the -X parameter to tell curl to use PUT. Identify the data you want to send to the server with the -d parameter followed by the data that you need to send to the server. Finally, you need to tell curl what URL to connect to:

IN ORDER FOR THIS INTERFACE TO WORK CORRECTLY AND CONSISTENTLY, IT IS IMPERATIVE THAT YOU CLEAN UP AFTER EACH EXECUTION OF THE SCRIPT.

```
url="http://ha.example.com/vars/alarmstate"
if grep -q 'Armed Stay' ~/tmp/results.tmp; then
  curl -X PUT -d value="ARMED Stay" $url
elif grep -q 'Armed Away' ~/tmp/results.tmp; then
  curl -X PUT -d value="ARMED Away" $url
```

```
elif grep -q 'Disarmed' ~/tmp/results.tmp; then
  curl -X PUT -d value=DISARMED $url
fi
```

When you put all of this together, the result is a block of code similar to this that will scan your file for the three string possibilities that will tell you the current alarm state. Because the evaluation is set up as an else/if, the if evaluation block will terminate when one of the expressions evaluates to true.

In order for this interface to work correctly and consistently, it is imperative that you clean up after each execution of the script. You may have noticed in the first block of code that data is appended to the temporary file using the >> I/O redirection operator. You append data, because the data streams into the file one line at a time. If you failed to use the append operator, the resulting file would contain only the last line of data from your message. Fortunately, cleaning up after ourselves is as easy as deleting our temporary file:

```
rm ~/tmp/results.tmp
```

Now that you have a script, it needs to be injected into the e-mail processing process in such a way that inbound mail is forced through the script. How this is accomplished will vary from system to system, so I won't go into great detail here, but I have implemented this very easily on both an open-source edition implementation of Zimbra and the e-mail platform provided by a very large and well-known US hosting provider. My current implementation resides with a hosting provider where I have an e-mail account called ha@example.com. I configured this account to "forward" all inbound mail to my script and then throw the message away.

AFTER HAVING MY INTERFACE SCRIPT IN PLACE FOR A NUMBER OF MONTHS, I HAD THE OPPORTUNITY TO USE THIS TROUBLESHOOTING METHOD.

With everything now in place, I configured my monitoring service to send alarm state change messages to my ha@example.com address and to my personal e-mail address. Having the messages go to both locations is helpful if you need to troubleshoot. Testing whether messages are coming from the monitoring company is as easy as checking my personal e-mail

to see if the state change message is present. If the message is present in my personal mail, but state change data isn't flowing to my home automation system, I then know that I need to troubleshoot my connectivity or my interface script itself.

After having my interface script in place for a number of months, I had the opportunity to use this troubleshooting method. I noticed that I was getting e-mail notifications in my e-mail, but state changes were not flowing to my home automation system. In order to verify that data was flowing to my script and that the data was being saved to my temporary file, I commented out the line that deletes the temporary file, then I forced an alarm state change. Sure enough, the file was created, but no change to the variable in my home automation system. In order to rule out a connectivity or a firewall issue, I then ran the curl command from the command line manually to see if my REST API call could reach my home automation system and change the variable. That worked fine. This conclusively proved that there was an issue with my interface script. At this point, I inspected the contents of the temporary file more closely, and I saw a new line in the headers that said "Content-Transfer-Encoding: base64". Apparently my monitoring company had made some changes to its e-mail system that I needed to account for. To do this, I would need to add a new block of code to see if the content of the newly arrived e-mail message was base64-encoded. If you find the message is encoded, use Perl and the the decode_base64 function in the MIME::Base64 module to decode the message:

```
if grep -q 'Content-Transfer-Encoding: base64' ~/tmp/results.tmp;
then perl -MMIME::Base64 -ne 'print decode_base64($_)'
<~/tmp/results.tmp>~/tmp/results2.tmp
  rm ~/tmp/results.tmp
  mv ~/tmp/results2.tmp ~/tmp/results.tmp
fi
```

Adding this block of code just before the block of code that performs the grep evaluations fixed the problem, and I was off to the races once again.

I have had this homebrew script interface in place for more than a year now. Other than the encoding issue, the interface's performance has been absolutely rock-solid. This allowed me to achieve the integration I wanted, even if that integration is a bit of a loose integration. Not only did this allow me to use the system that I already had, but also the implementation was completely free, and after all, who doesn't like *free*?■

James Litton has held executive-level positions at numerous organizations and is a partner and CEO of Identity Automation.