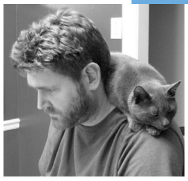
DAVE JOSEPHSEN

iVoyeur: *you* should write an NEB module, revisited



David Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper Award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.

dave-usenix@skeptech.org

IN THE LAST ISSUE I SPENT SOME TIME trying to get you interested in writing Nagios Event Broker (NEB) modules. NEB modules, as you no doubt recall from the literary triumph that was my last article [1], are small, user-written shared object files that can extend or change the functionality of Nagios. If you dislike something about how Nagios functions or wish a hook had been added for your favorite monitoring tool, NEB modules are for you. In fact, if you use Nagios at all and have written any code related to your install that isn't a plug-in, then NEB modules are for you too. Heck, if you haven't already flipped the page looking for something more interesting to read, then you should be writing NEB modules.

So, in a rare (for me) fit of long-term attention span, this month I want to follow up on a few things I didn't get to cover in the last article. I took a few moments of time to get my Nagfs module working with the 3.x series of Nagios, and I also wanted to give you some hard examples of how the new custom external commands feature in the 3.x series can be used by an NEB to do interesting things.

In fact, when I said I took a few moments of time to get my Nagfs module working in 3.x, I literally meant a few moments. No code needed to be changed at all. The only hiccup I ran into was that the 3.x series of Nagios includes the glib libraries if your module has NSCORE defined, as mine does, and glib on my system was in a non-obvious place. So to port my module from 2.x to 3.x I went from typing:

```
gcc -shared -o nagfs.o nagfs.c
```

to typing:

```
gcc -shared -I/usr/include/glib-1.2 -o nagfs.o nagfs.c
```

Since I brought it up, I should probably write a word or two about the NSCORE compiler definition. In terms of an executive summary, I can tell you that I don't really know why it's there, but your module gets a bunch more information if you define it, so I do. If you're curious, you can take a look at the module/helloworld.c file in the base directory of the Nagios tarball and note that it is not set, so it is not in fact required for your module to operate. I would then direct your attention to the `service_struct` definition in the include/objects.h in the base directory of the tarball. Note that all the really great stuff you'd probably want to know is only available if NSCORE is defined. This is true of pretty much all the interesting structs, so I define it. The includes/config.h file in the base directory of the Nagios tarball includes glib if NSCORE is defined.

To make a long story short, despite the fact that a lot of NEB-related code was touched in the move from 2.x to 3.x, simple modules like mine will probably not need to be changed to compile, which is happy news and hints at solid engineering. Three cheers for those crafty Nagios developers!

On to the custom external commands feature, beginning with a short definition. Nagios can be controlled by passing commands into a FIFO called the external commands file. External commands are the preferred way to change Nagios's runtime settings from external scripts. For example, if you needed to schedule downtime for a large number of individual hosts, you could write a script that generated input to the external command file rather than using the cumbersome Web interface. The most common use of external commands is probably implementing passive host and service checks [2].

External commands have the syntax:

```
[time] command_id;command_arguments
```

The square brackets are literal and time is in epoc seconds format, for example:

```
[1222309414] foo;bar
```

The commands themselves are statically defined, and each command takes different numbers and types of arguments. These are documented at the Nagios Web site [3]. New to Nagios 3.0, and the point of this ramble, are custom commands. Simply preface the command name with an underscore and Nagios will treat the command as "custom." Custom commands are not defined and may contain as many freeform arguments as you wish. (Well, there's probably a buffer-size cap somewhere, but I've never hit it.) So although this example will be ignored completely by Nagios, the following command will be parsed as a custom command and passed by the event broker to any modules that are interested in receiving it:

```
[1222309414] _foo;bar

/* handle data from Nagios daemon */
int nagfs_handle_data(int event_type, void *data){
  nebstruct_service_status_data *ssdata=NULL;
  nebstruct_host_status_data *hsdata=NULL;
  nebstruct_external_command_data *exdata=NULL;
  service *svc=NULL;
  host *hst=NULL;
  char temp_buffer[1024];

  /* what type of event/data do we have? */
  switch(event_type){

  case NEBCALLBACK_SERVICE_STATUS_DATA:
  //service status data occurs every time a check runs. We use this to update the service file
  in each host's directory.

     ssdata=(nebstruct_service_status_data *)data;
     //ss data gives us a pointer directly to the service object
     svc=ssdata->object_ptr;
     nagfs_write_service_status(svc->host_name, svc->description, svc->current_state, svc->state_type );

   break;

  case NEBCALLBACK_HOST_STATUS_DATA:
  //service status data occurs every time a host check runs. We use this to update the HOST file.
```

```
    hsdata=(nebstruct_host_status_data *)data;
    //ss data gives us a pointer directly to the service object
    hst=hsdata->object_ptr;
    nagfs_write_host_status(hst->name, hst->current_state, hst->state_type );

  break;

case NEBCALLBACK_EXTERNAL_COMMAND_DATA:
//external commands are user-submitted commands from the external command file

    exdata=(nebstruct_external_command_data *)data;
    //the external command struct doesn't give us a pointer to a command struct
    snprintf(temp_buffer,sizeof(temp_buffer)-1,"Nagfs: got command: %s",exdata->command_string);
    temp_buffer[sizeof(temp_buffer)-1]='\0';
    write_to_all_logs(temp_buffer,NSLOG_INFO_MESSAGE);

    if((strcmp(exdata->command_string,"_nagfs_die")) == 0) {
        nebmodule_deinit(0,0)
    }

  break;

default:
    snprintf(temp_buffer,sizeof(temp_buffer)-1,"nagfs: just got some unknown data. Weird.." );
    temp_buffer[sizeof(temp_buffer)-1]='\0';
                write_to_all_logs(temp_buffer,NSLOG_INFO_MESSAGE);

  break;

  }

return OK;
  }
```

LISTING 1

Listing 1 is a modified version of my event handler function from last month's article. Not shown in the listing is the extra registration call we must add to the init function to begin receiving external command events:

```
neb_deregister_callback(NEBCALLBACK_EXTERNAL_COMMAND_DATA,nagfs_handle_data);
```

As you probably recall from the last article, the first argument is a constant that defines what we want to register for, and the second argument is a function pointer back to our own event handler function. I'll save you a grep or two by pointing out that the event-type constants are defined in includes/nebcallbacks.h in the base of the Nagios tarball. As with everything in the Nagios source, the names are self-explanatory and easy to find. The struct that makes up an external command is defined in include/nebstructs.h. I'll paste it into Listing 2, so I can briefly discuss it here.

```
typedef struct nebstruct_external_command_struct{
    int                 type;
    int                 flags;
    int                 attr;
    struct timeval      timestamp;

    int                 command_type;
    time_t              entry_time;
    char                *command_string;
    char                *command_args;
      }nebstruct_external_command_data;
```

LISTING 2: AN EXAMPLE OF THE EXTERNAL COMMAND STRUCT

The external command struct in Listing 2 is a bit different from the service and host data structs we dealt with in the last article. In the latter structs we were given a pointer to an object that represented the actual service or host in memory. The external command struct, however, refers to no other Nagios object, because there is no other object to refer to. Everything you need to know about the custom command—its execution time, name, and arguments—can be gotten directly from the struct passed to us from the broker. The variable names that contain these pieces of information are, of course, self-explanatory.

Aside from a few variables at the top of the function in Listing 1, all I've added is a case to the `switch` loop that logs the name of the command we've received from the broker. If the event name matches `_nagfs_die`, then the module commits suicide by calling its own `deinit` function.

If you need to talk to your NEB module from an external source such as a script or another server, custom external commands are the perfect means to do it. Here's an example: Imagine a VRRP-like protocol for Nagios fail-over servers. Both servers send each other "I'm alive" messages, and whoever has the highest priority is the master. All service checks and notifications on the backup server are suppressed while the master is alive. This could be implemented in NEB as a single piece of code; that is, all participating servers would run exactly the same NEB module. Best of all, the message-passing interface (the hard part) is already written for you in the form of existing plug-ins plus the custom external commands feature.

In versions of Nagios prior to 3 you would have had to implement your own mechanism for message passing, and that would have meant scheduling events for your module to wake up and check for messages, so this feature makes NEB modules much easier to write and should hopefully inspire you to write modules that do even cooler things. That last example I just made up off the top of my head; the thought of what problems the LISA crowd could solve with NEB makes me all giddy.

So, beloved *;login:* readers, I sincerely hope these two articles have at least piqued your interest in the Nagios event broker. If you use Nagios and need customizations, please, please, please write an event broker module. They're fun to write, and really I'd like to reap the benefit of your hard work, because, let's face it, you're smarter than I am and I couldn't afford your consulting fee anyway.

Take it easy.

REFERENCES

[1] My last article: http://www.usenix.org/publications/login/2008-10/pdfs/josephsen.pdf.

[2] Passive service checks in Nagios: http://nagios.sourceforge.net/docs/3_0/passivechecks.html.

[3] List of Nagios external commands: http://www.nagios.org/developerinfo/externalcommands/commandlist.php.