Contributing Plugins for Uptime Infrastructure Monitor

Introduction

With the introduction of the Extension Manager in up.time 7.3, plugins and gadgets are now even easier to install. As part of this change, we have also posted the source code for all plugins on github. Use these examples as starting templates for building your own plugins.

Plugins

What's a plugin?

Plugins are custom service monitors that allow Uptime Infrastructure Monitor to monitor and collect metrics from a variety of different sources. In most cases they're a combination of XML which defines the input and output variables, and some simple shell scripts that handle the actual task of monitoring. These monitors usually consists of some sort of login or connectivity check, as well as exposing key metrics in a format that can be understood by Uptime Infrastructure Monitor. The specifics of the scripts themselves depend on what protocol and libraries are required to the talk to device/system/service on the other end. But this doc will explain howto use element details from Uptime Infrastructure Monitor to control where these scripts are targeted. As well howto define your own input variables that are stored as service monitor settings in the Uptime Infrastructure Monitor.

Inputs Variables

Uptime Infrastructure Monitor uses Environment Variables as it's main way of exposing information to all custom scripts. The most important of these environment variables is `UPTIME_HOSTNAME` which provides the hostname of the element the service monitor is running against. Next comes the various input variables from the monitor's xml file, which provide the specific details needed for the script itself. The best way to understand how this works is to take a look at an example of the xml definition a monitor, how that looks as service monitor settings in the GUI, and finally how all this information ends up as environment variables in the script.

A good example of this is the 'Custom Remote Monitor' as this plugin also provides a great starting point for custom monitors that also rely on a an agent side script. But for now we're just going to focus on the XML details and the script that runs directly on the monitoring station itself. The full contents of the xml file are available on github here .

XML - Grid Info

The first section of the XML file contains details about the name of the plugin/service monitor and which category its shows-up under on the 'Add Service Monitor' screen.

```
<?xml version="1.0" encoding="UTF-8"?>
<uptime xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.uptimesoftware.com
/erdc/erdc erdc" xmlns="http://www.uptimesoftware.com/erdc/erdc" xmlns:upt="http://www.uptimesoftware.com/erdc
/erdc" version="4.0" target_grouping="other">
        <class>Custom Remote Monitor</class>
        <exec>com.uptimesoftware.uptime.erdc.MonitorCustomScriptRunner</exec>
        <erdc_version>3.01</erdc_version>
        <category>Advanced and Script Monitors</category>
        <category_related>Applications - General</category_related>
        <grid_info>
                <supported_versions>7.3</supported_versions>
                <supported_platforms>windows, linux</supported_platforms>
                <upgrade_in_place>true</upgrade_in_place>
                <requires_agent_scripts>false</requires_agent_scripts>
        </grid_info>
```

Let's talk about each of these tags, and what they represent:

- `<class>` - This is the service monitor's name as seen on the 'Add Service Monitor' screen in the Uptime Infrastructure Monitor GUI.
- `<exec>` - This tells Uptime Infrastructure Monitor that our plugin is a `MonitorCustomScriptRunner .` There are also java type plugins, but this doc will only focus on Custom Scripts.
- `<erdc_version> -` This is the version number of the plugin which is used to identify when new versions are available.
- `<category>` - This is the primary category on the 'Add Service Monitor' screen where you'll be able to find this monitor.
- `<category_related>` This is a secondary category where the monitor will be displayed in the 'Related Monitors' list.
- The `<grid_info>` section outlines details about which OS platforms this monitor can run on, as well as which versions of Uptime Infrastructure Monitor. Which is used by the Extension Manager to filter which plugins to show as available for installed.
    - `<supported_versions>` - This is the version of Uptime Infrastructure Monitor monitoring station this plugin is compatible with.
    - `<supported_platforms>` - This is a list of which monitoring station platform a monitor can run on. This can either windows or linux, or both like in the example above.
    - `<upgrade_in_place>` - This tells Uptime Infrastructure Monitor that the plugin can be updated without having to re-define the services monitors assigned to it. In most cases you'll want to set this to true.

Next comes the elements section of the XML file. This is a list of element tags which define the input and output variables you see when editing a service monitor in the Uptime Infrastructure Monitor GUI. The contents of these variables are stored in the Uptime Infrastructure Monitor datastore just like regular service monitors.

XML - Elements

The first of these elements are the process_linux & process_windows which control which script actually gets run on the monitoring station when a service monitor is triggered.

In most of the plugins created by Uptime Infrastructure Monitor these will point to either a .bat or a .sh script depending on the OS type. These wrapper scripts then call a php script that does the actual monitoring. This particular setup is to make it easier to support both Windows and Linux with the same script. As well minimize the external dependencies on things not bundled into the default install of Uptime Infrastructure Monitor. Your free to use whatever scripting languages you want here, depending on what's best suited for the task at hand.

```
        <elements>
                <element name="process_linux" parameter_type="input" data_type="string" gui_basic="1"
range_type="0" hidden="true">
                        <control_options> size:40 </control_options>
                        <default_value>plugins/scripts/monitor-custom-remote/monitor-custom-remote.sh<
/default_value>
                        <short_description>Script Name</short_description>
                        <long_description>Name of the script/binary/process to be executed by up.time <
/long_description>
                        <validation_rule>
                                <alphanumeric/>
                        </validation_rule>
                        <error_message>
                                <alphanumeric>required field</alphanumeric>
                        </error_message>
                        <gui_type>
                                <string/>
                        </gui_type>
                </element>
                <element name="process_windows" parameter_type="input" data_type="string" gui_basic="1"
range_type="0" hidden="true">
                        <control_options> size:40 </control_options>
                        <default_value>plugins/scripts/monitor-custom-remote/monitor-custom-remote.bat<
/default_value>
                        <short_description>Script Name</short_description>
                        <long_description>Name of the script/binary/process to be executed by up.time <
/long_description>
                        <validation_rule>
                                <alphanumeric/>
                        </validation_rule>
                        <error_message>
                                <alphanumeric>required field</alphanumeric>
                        </error_message>
                        <gui_type>
                                <string/>
                        </gui_type>
                </element>
```

Full details about what the various attributes of the XML file's element tag control are available here: 'Integration Guide -> Plugin Guide' . But for now, we'll just cover the most important details from the example above:

- `name` - This is the name of variable itself. In the examples above you can see this is either set to process_windows or process_linux . You need to have at least one of these defined for every plugin.
- `parameter_type` - This can be either input or output. Input parameters will be set as an environment variable at run time. We'll talk more about output type parameters below.
- `data_type` - This can be either string or integer. This usually more important on the output variable side of things, as it also controls what WARN /CRIT threshold comparisons can made against it.
- `hidden` - This can either be true or false . This controls whether or not a parameter is displayed on the Edit Service Monitor Screen. A hidden variable can not be edited by a user in the GUI, but can still be updated in the db directly.
- `<default_value>` - This provides the default value for a variable. Which is especially important for the examples above, as both are set as hidden and are not editable from the GUI.

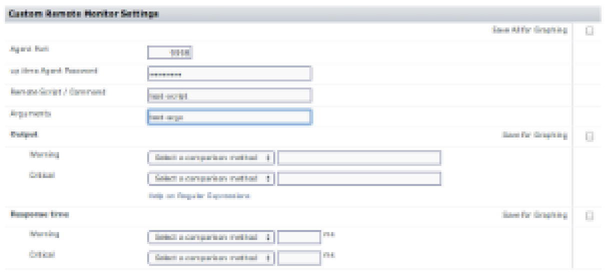
Now that we've looked at the process_windows and process_linux fields, let's take a look at some 'input' variables for this monitor. If your familiar with the 'Custom Remote Monitor' in Uptime Infrastructure Monitor, you've likely seen this combination of Agent Port, Agent Password, Remote Script fields before, as these are commonly used to trigger custom agent scripts.

- We can see the 'Agent Port' parameter is setup as type integer, and has the default port 9998 already set for us.
- The password parameter is setup as a data_type of string, but also has the gui_type set to <password /> which as mentioned above, tells Uptime Infrastructure Monitor to encrypt this value when it's stored in the database, as well as blank out the field itself when displayed in the GUI.
- The remote_script & args fields are both setup as basic string fields with nothing special about them.

```
                <element name='port' parameter_type='input' data_type='integer'
                gui_basic='1' range_type='0' units=''>
                        <control_options>size:8</control_options>
                        <default_value>9998</default_value>
                        <short_description>Agent Port</short_description>
                        <long_description>up.time Agent Port (default is 9998)</long_description>
                        <validation_rule/>
                        <error_message/>
                        <gui_type>
                                <integer/>
                        </gui_type>
                </element>
                <element name='password' parameter_type='input' data_type='string'
                gui_basic='1' range_type='0' units=''>
                        <control_options>size:8</control_options>
                        <default_value></default_value>
                        <short_description>up.time Agent Password</short_description>
                        <long_description>Password setup on the agent system</long_description>
                        <validation_rule/>
                        <error_message/>
                        <gui_type>
                                <password/>
                        </gui_type>
                </element>
                <element name='remote_script' parameter_type='input' data_type='string'
                gui_basic='1' range_type='0' units=''>
                        <control_options>size:8</control_options>
                        <default_value></default_value>
                        <short_description>Remote Script / Command</short_description>
                        <long_description>Script name (posix) or command (win) setup on the agent<
/long_description>
                        <validation_rule/>
                        <error_message/>
                        <gui_type>
                                <string/>
                        </gui_type>
                </element>
                <element name='args' parameter_type='input' data_type='string'
                gui_basic='1' range_type='0' units=''>
                        <control_options>size:8</control_options>
                        <default_value></default_value>
                        <short_description>Arguments</short_description>
                        <long_description>Arguments that will be sent to the remote script</long_description>
                        <validation_rule/>
                        <error_message/>
                        <gui_type>
                                <string/>
                        </gui_type>
                </element>
```

Edit Service Monitor View:

Now that we've talked about the details that go into a plugin's XML file, let's take a look at how these input variables are displayed in the Uptime Infrastructure Monitor GUI when creating an instance of the 'Custom Remote Monitor':



Custom Remote Monitor Settings

Agent Port: 9998
up.time Agent Password: ********
Remote Script / Command: test-script
Arguments: test-args

- The first thing you'll notice is that we don't see either the process_windows or process_linux fields, but that's to be expected here, as those fields where set as hidden.

- The contents of the password field are obscured as expected.
- The Remote Script & Arguments fields are basic string fields.
- You can also see the two output values for this monitor, but we'll talk about those more later.

Environment Variables

Next let's take a look at how all these parameters and input fields come together as the environment variables that are used by the script itself. Environment variables aren't normally/displayed or retained by the service monitor when it runs, but they're easy to capture for testing/debugging purposes by adding the appropriate command to your shell/bat script:

- Linux: `printenv > /var/tmp/envvars.txt`
- Windows `set > C:\envvars.txt`

There's normally plenty of other environment variables that are set, but we've filtered those out to only look at the uptime values that are coming from our service monitor's XML definition.

```
UPTIME_ARGS=test-args
UPTIME_TIMEOUT=60
UPTIME_SYSTEM_TYPE=NODE
UPTIME_PORT=9998
UPTIME_PROCESS_WINDOWS=plugins/scripts/monitor-custom-remote/monitor-custom-remote.bat
UPTIME_PROCESS_LINUX=plugins/scripts/monitor-custom-remote/monitor-custom-remote.sh
UPTIME_REMOTE_SCRIPT=test-script
UPTIME_HOSTNAME=cat1.uptimesoftware.com
UPTIME_PASSWORD=test-pass
```

Here's a few things you should notice about these:

- They all start with `UPTIME_` and the contents of the name attribute from the element tag.
- The `UPTIME_HOSTNAME` has the hostname of the element the monitor is running against.
- The rest of our input variables show-up as expected with the same contents as we set in the GUI example above.

Shell/Bat Wrapper Scripts

Now let's take a quick look at what the `custom_remote.bat` & `custom_remote.sh` scripts actually do with these environment variables. As mentioned above, most of the Uptime Infrastructure Monitor provided plugins rely on a common PHP script to perform the actual task of monitoring. As such the bat & sh scripts are just responsible for telling Uptime Infrastructure Monitor's bundled php executable to run the required script, pass along the environment variables in the correct order. The specifics of how this done obviously varies between both Windows and Linux, so let's take a look at both methods.

See monitor-custom-remote.bat on github:

```
@echo off
..\..\..\apache\php\php.exe rcs.php -h=%UPTIME_HOSTNAME% -p=%UPTIME_PORT% -s=%UPTIME_PASSWORD% -c=%
UPTIME_REMOTE_SCRIPT% -a=%UPTIME_ARGS%
```

From the example here, you can see that windows allows you access environment variables by wrapping the name in % characters, ie `%UPTIME_HOSTNAME%` . So we've used that to arrange the input variables from xml into the order expected by the rcs.php script. You may also notice the `..\..\..\apache\php\php.exe` path used to start php. Whenever a script monitor is triggered the current working directory is set to the location of the script itself, which is in this case is `'C:\Program Files\uptime software\uptime\plugins\scripts\monitor-custom-remote\monitor-custom-remote.bat'` But the location of the uptime directory can vary depending on where it was initially installed. So we use a relative path from the plugin's location to find `<uptime_dir>\apache\php\php.exe .` When making your own plugins, you can usually simplify this by using the hardcoded path of your uptime directory.

See monitor-custom-remote.sh on github:

```
#!/bin/sh


inst=`grep ^inst /etc/init.d/uptime_httpd | cut -d= -f2`
MIBDIRS=$inst/mibs
export MIBDIRS


/usr/local/uptime/apache/bin/php rcs.php -h=$UPTIME_HOSTNAME -p=$UPTIME_PORT -s=$UPTIME_PASSWORD -
c=$UPTIME_REMOTE_SCRIPT -a=$UPTIME_ARGS
```

In the case of the linux shell script, we can access the contents of our environment variables by prefacing the name with the $ symbol. We then pass the same input variables along to php in the same order as in the bat script above. Just like on Windows the current working directly is set to the location of plugin script itself, but in this case we can use the contents of the /etc/init.d/uptime_httpd script to find the location the Uptime Infrastructure Monitor install directory and use to that to set the location of an extra MIBDIRS environment. This only required for PHP scripts on linux, so that the php_snmp library doesn't complain about being unable to find it's mibdirs. If your not using php in your own script, you can ignore this completely. The other thing worth noticing here is that we've hardcoded the path of Uptime Infrastructure Monitor's bundled php executable, as uptime's Apache folder is always installed into the default location of /usr/local/uptime/apache even if Uptime Infrastructure Monitor was installed elsewhere, this prevents us from using the same relative path method as we did on Windows.

We won't get into the details of the rcs.php script itself as this will typically change depending on the details of the plugin itself. But the full contents of this file on github here if your interested.

Output Variables

Now that we've looked at all the details that go into a plugin scripts input variables. All the way from the xml file, the Edit Service Monitor Screen, and how they finally get exposed to the custom script as environment variables. Let's take a look at how a custom monitor can return it's output back to Uptime Infrastructure Monitor, and the different things we can do it with.

XML - Output Variables

Just like input variables, output variables are also defined in the xml file via the same element tags. They're covered by the same basic rules outlined above, and full details for their XML definitions are available here: 'Integration Guide -> Plugin Guide' .

Below is the two output variables that are defined in the Custom Remote Monitor's xml file. For reference the full version of this file is available on github he re .

```
                <element name='custom1' parameter_type='output' data_type='string'
                gui_basic='1' range_type='0' units=''>
                        <control_options>size:8</control_options>
                        <default_value></default_value>
                        <short_description>Output</short_description>
                        <long_description>Output from the script</long_description>
                        <validation_rule/>
                        <error_message/>
                        <gui_type>
                                <string/>
                        </gui_type>
                </element>
                <element name="timer" parameter_type="output" data_type="integer" gui_basic="0"
                        range_type="0" units="ms">
                        <control_options>size:8</control_options>
                        <default_value/>
                        <short_description>Response time</short_description>
                        <long_description>Response time in ms</long_description>
                        <validation_rule/>
                        <error_message/>
                        <gui_type>
                                <integer/>
                        </gui_type>
                </element>
        </elements>
        <help>Execute remote scripts via the up.time agent. Output from the remote string is returned as String
Data.</help>
</uptime>
```

Here's some things you notice about these examples:

- Both cases, the parameter_type is set as output . This tells uptime that this is output we expect to see back form the script.
- There's an optional unit attribute in the element tag, which is used to show that the response time is being stored in 'milliseconds' . This unit will also be displayed when graphing these metrics within Uptime Infrastructure Monitor.
- We have an example of both the integer & string data_types, which in the case of output variables controls what type of threshold comparisons can be used to set the monitors WARN/CRIT status. Integer type values allow for Greater Than/Less Than numerical comparisons. Where-as strings have contains/does not contains and regex style comparisons.

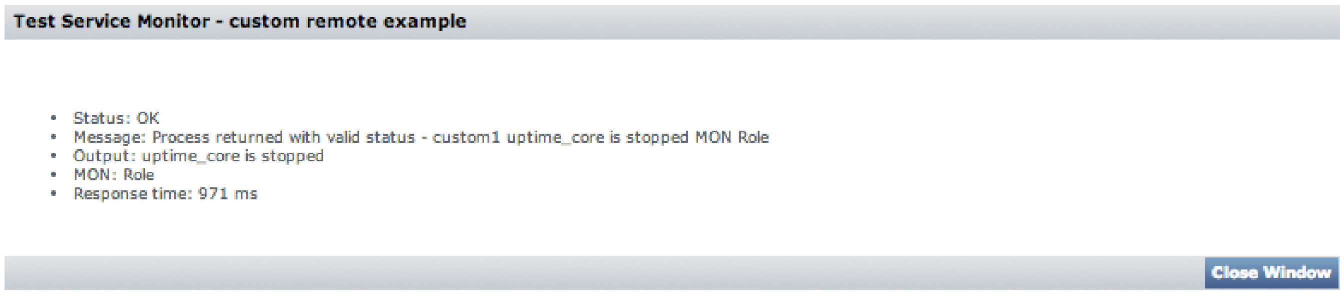
Now that we've see what the output variables look like in the xml itself, lets take a look at what the script needs to return so that Uptime Infrastructure Monitor knows which output variable is which. This based on the element tag's name attribute. Which is used to look for a line of output from the script that starts with that name, which in the case above is 'custom1'. The rest of that line will then be pulled into Uptime Infrastructure Monitor as the variable and can used for threshold comparisons. In our example here, this custom1 variable will be treated as string. In monitors with multiple output variables, the output from the script itself can come any order, but any lines that don't start with the name of a variable will be ignored.

The 'timer' output variable is a special case that will always contain the time it took the service monitor to run in milliseconds. This doesn't need to be provided as part of the output from your custom script.

The best way to understand how this works it to look at the output on the 'Test Service Monitor' screen. Which will show you the raw output from the service monitor, and then a list of any output variables that Uptime Infrastructure Monitor was able to parse out based on their names. For an example here, we'll use the 'custom remote monitor' with an agent side script called 'check-uptime' which returns output similar to:

```
uptime_core is stopped
 MON Role
```

When this is triggered via the 'Test Service Monitor' button, you would see something like this:



- Status: OK
- Message: Process returned with valid status - custom1 uptime_core is stopped MON Role
- Output: uptime_core is stopped
- MON: Role
- Response time: 971 ms

The first thing you'll likely notice here is that the Monitor has set it's Status to OK , which is the default state for a service monitor that's run successfully, as long as none of it's threshold comparisons are triggered. The 'message' line shows use that the process returned a valid status , which is then followed by the output from the plugin monitor. ie. ' custom1 uptime_core is stopped MON Role'

You may wonder where the 'custom1' string is coming from as it's not in the example output of the agent side script above. This is inserted by the rcs.php and used to mark that the output from the agent script should link up with the 'custom1' variable from the XML. Which you can see on the next line 'Output' where 'custom1' has been replaced with that variable's Short-Description also form the xml. The next line is an example of Uptime Infrastructure Monitor attempting parse the second line of output from our agent script into a variable called 'MON' but because this isn't defined in our xml file, it won't be available for comparisons within the GUI. The final line here is the Response time value, which is special timer mentioned above.

The general take-away from this test service monitor example is that when Uptime Infrastructure Monitor is able to parse/understand the output from a script, the matched up variables will be listed line by line after the message. With the 'name' of the variable replaced by it's short description. If you don't see your output variable listed like this, then double check the names of your output variables and how the script itself is returning its output.

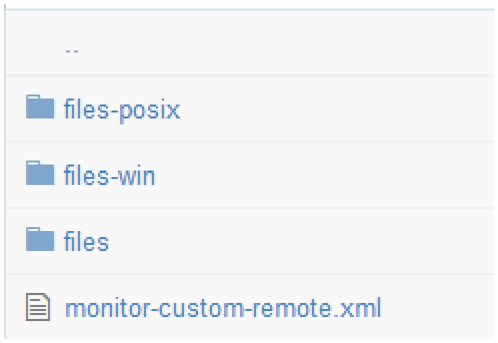
Installing Plugins

Now that we've seen how plugin monitors are made of a combination an XML file that defines the various inputs/outputs and a script performs the actual task of monitoring, Let's take a look how these pieces are packaged together, and how Uptime Infrastructure Monitor knows which plugins are available for install.

Packaging Plugins

The best way to understand how Uptime Infrastructure Monitor plugins are packaged is to take a look at the zip file for a plugin itself. We'll use the same 'Custom Remote Monitor' from above as our example again. The actual zip file that is used when installing the plugin is located within this repo's 'dist' folder, along with some .upk files (The legacy plugin format used by older versions of Uptime Infrastructure Monitor).

You can easily download the 'monitor-custom-remote.zip' yourself, and look at the contents directly, but the contents of that archive are just a zipped up version of the repo's 'pkg' directory, which you can easily view in github's web interface here.



..
files-posix
files-win
files
monitor-custom-remote.xml

The contents of the zip/pkg folder are made up of the xml file that we've already discussed in detail above, and a mix of one or more 'files' folders. These folders contain the actual scripts that will be extracted into place on the Uptime Infrastructure Monitor monitoring station. The reason for the files-win & files-posix folders are that plugins may require different files or scripts depending which OS (ie. Windows or Linux) the Uptime Infrastructure Monitor monitoring station is running. As such the contents of the 'files-win' folder will only be extracted on Windows monitoring station, and the 'files-posix' folder will only be extracted on a Linux monitoring station.

As mentioned before, most of the plugins created by Uptime Infrastructure Monitor are intended for cross platform use. This is why is most of them they have all three of these 'files' folders. When creating your own plugin monitor, you can focus just on the OS platform of your Uptime Infrastructure Monitor monitoring station, and simplify the packaging of your plugin by placing all the scripts within the 'files' folder.

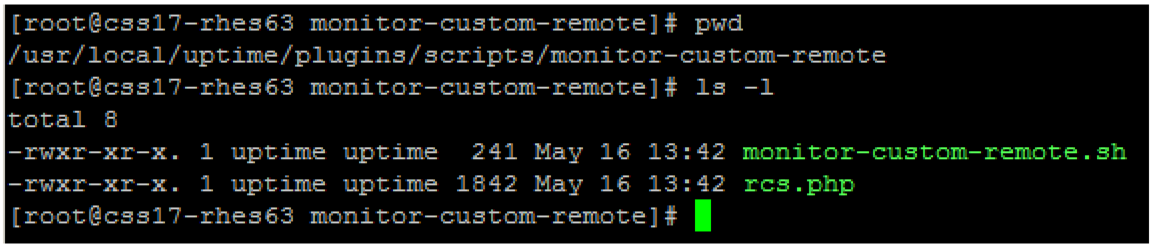
The mechanism for how these 3 folders are extracted during the plugin's installation are the same general process whether the monitoring station is running Windows or Linux. Which is that each of the 'files' folders will be extracted into the directory where Uptime Infrastructure Monitor is installed on the monitoring station. Any sub-folders within the plguin's 'files' folders will be created along the way if they don't already exist. Though in most cases, you'll want to place all your plugin's scripts/files within it's own sub-folder in `uptime_dir/plugins/scripts/` .

Another thing you may notice with some of the plugins created by uptime is that they have scripts that go in both the `uptime_dir/plugins/scripts/` and `uptime_dir/scripts/` folders. This is because the 'scripts' folder was the legacy location for storing plugin related files/scripts on older versions of Uptime Infrastructure Monitor. The duplication here is to

When creating your own plugins, you should be placing all your scripts/files within your own folder in `uptime_dir/plugins/scripts/`

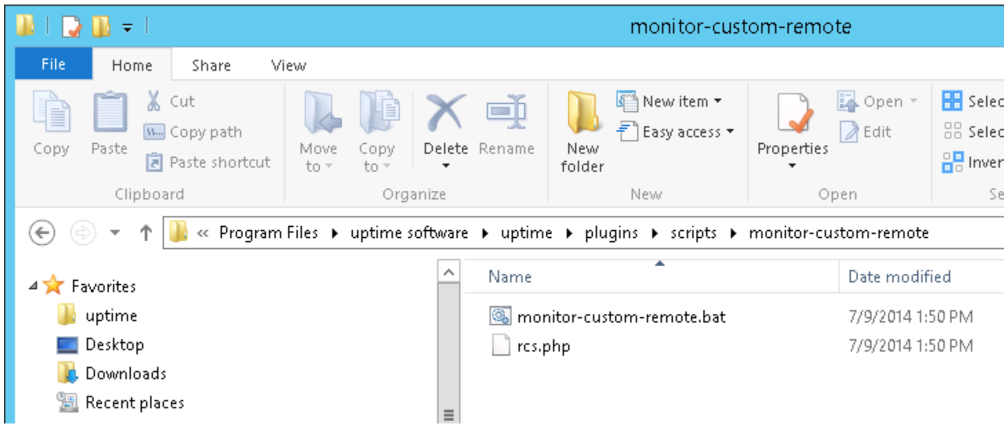
So in the case of the Custom Remote Monitor's 'files' folder, you'll notice it contains some sub folders (ie. plugins/scripts/monitor-custom-remote) which eventually contain a single rcs.php file.

When installed on a Linux monitoring station those files will end up in a location similar to below. (In this example, Uptime Infrastructure Monitor was installed in the default /usr/local/uptime location). The 'monitor-custom-remote.sh' file comes from a similar sub-folder within the 'files-posix' folder.



```
[root@css17-rhes63 monitor-custom-remote]# pwd
/usr/local/uptime/plugins/scripts/monitor-custom-remote
[root@css17-rhes63 monitor-custom-remote]# ls -l
total 8
-rwxr-xr-x. 1 uptime uptime  241 May 16 13:42 monitor-custom-remote.sh
-rwxr-xr-x. 1 uptime uptime 1842 May 16 13:42 rcs.php
[root@css17-rhes63 monitor-custom-remote]#
```

Similarly on a Windows monitoring station that file would end up in the location similar to below. (In this example Uptime Infrastructure Monitor was installed into the default C:\Program Files\uptime software\up.time\ location for Windows). The 'monitor-custom-remote.bat' file comes from a similar sub-folder within the plugin's 'files-win' folder.



« Program Files ▸ uptime software ▸ uptime ▸ plugins ▸ scripts ▸ monitor-custom-remote

Name	Date modified
monitor-custom-remote.bat	7/9/2014 1:50 PM
rcs.php	7/9/2014 1:50 PM

Extension.json

Now that we've seen how a plugin monitor is packaged into it's own zip file,the last piece we need to look at the extension.json file that the monitoring station uses to know which plugins are available for install and where to get the associated zip files. By default Uptime Infrastructure Monitor will attempt to get these details from this location: http://the-grid.uptimesoftware.com/extension.json on our website.

In order to install your own plugins for testing, you'll need to create your own copy of this file that includes details about where to locate your own plugin files. The general process for that is outlined in this KB article: 'How To Install Plugins Without Direct Internet Access' . Though that article focuses on howto internally mirror the currently available plugins for environments where the Uptime Infrastructure Monitor server doesn't have direct access to the external internet. In our situation, we instead need to use the same general process to create our own extension.json file that lists details about our own plugins.

Each plugin listed will need a entry in the json file similar to the below example of the 'Custom Remote Monitor':

```
{
        "category": "Advanced and Script Monitors",
        "supported_versions": [
                "7.3"
        ],
        "name": "Custom Remote Monitor",
        "git_file_target": "https://github.com/uptimesoftware/monitor-custom-remote/raw/master/dist/monitor-
custom-remote.zip",
        "icon_url": "http://the-grid.uptimesoftware.com/img/posts/placeholder-logo.png",
        "upgrade_in_place": true,
        "requires_additional_setup": false,
        "requires_agent_scripts": false,
        "version": "3.01",
        "supported_platforms": [
                "windows",
                "linux"
        ],
        "grid_url_target": "http://the-grid.uptimesoftware.com/plugin/monitor-custom-remote.html",
        "type": "plugin",
        "description": "Execute remote scripts via the up.time agent. Output from the remote script is returned
as String Data"
}
```

Most of these fields are pretty self explanatory and match up with similar fields from the monitor's XML file. The most important of these fields is `git_file_target` which tells Uptime Infrastructure Monitor's extension manager where to find the zip file it needs to download to install the plugin. Most of the plugins provided by Uptime Infrastructure Monitor have this field pointing to a location on github.com, though that isn't required, and you can place these files anywhere that is accessible from your monitoring station over http.

The next most important fields for a plugin are the 'name' & 'version' as these are used to figure out which plugins are currently installed within Uptime Infrastructure Monitor and if they're running the current version. A combination of these two fields controls whether a user will see the Installed/Install/Upgrade buttons within the extension manager. The contents of the name field in the JSON needs to exactly match the XML's class tag.

Contributing a Plugin

By now you've hopefully been able to create your own plugin monitor for use with Uptime Infrastructure Monitor and are asking yourself how you can share your creation with other Uptime Infrastructure Monitor users.

Choosing a License

One thing you may not of thought about much more when creating software/scripts for internal project is 'What Copyright License should I release this under?'

If your not aware, there's a variety of different copyright licensing models available for open source software that outlines what changes someone can make to your code and how can they share it among others. At Uptime Infrastructure Monitor we recommend using one of the 'Creative Commons' Licences as they provide easy to understand descriptions of what the license covers. Details about these licenses are available at: https://creativecommons.org/licenses/

Of the licenses listed on the Creative Commons website, we recommend using the Attribution-ShareAlike with plugins as it provides the best fit for how plugins are typically created and expanded upon. Once you've chosen the appropriate license for your plugin, you'll need to mention it somewhere within the plugin itself, typically either as part of the readme itself or by adding a license.txt file to your plugin repo. See this page on github for some common ways people handle this.

Putting your plugin on github

If you haven't been using git or some other version control tool while creating your plugin then it's time to create a repository for it on github, as that's the easiest way to share your creation with us and make it available for download via the Extension Manager.. Here's some documentation on github.com that covers the basics of howto create your own repository and a general introduction for working with git.

Once your plugin is available on github it's time to contact uptime-support@idera.com and let us know about your creation. From there we'll take a look at what your plugin does, and how it works, and do some of the remaining tasks involved in making it available for download via Uptime Infrastructure Monitor's extension manager.