GDMA Advanced

Overview

This page reference is designed for those that have used GDMA previously, have it already set up and working, and would like to use the advanced features.

CONTENTS

RELATED RESOURCES

- Configuring Auto Registration

WAS THIS PAGE HELPFUL?

- Leave Feedback

1.0 Detailed Installation and Setup

Controlling poller and spooler status messages

The GDMA profiles contain services called `gdma_poller` and `gdma_spooler`. By default, these services will receive status messages from the poller and spooler processes. These messages tell you about how much time the poller is using to perform the checks, and whether that time is excessive. The spooler reports transmission failures, and statistics about how many messages it sent, and if any messages were purged, etc. The state of the services will change, as well, based on whether there is an error to report.

These messages are useful, but can be excessive, especially in large configurations. They should be left on in initial deployments, but after calibration is done, you may wish to curtail their use. or even remove the services. *GDMA* adds some parameters to allow you better control over these messages:

- **Poller_Service="gdma_poller"** - This parameter controls the name of the service to send poller messages under, defaulting to `gdma_poller`. You can change this to another service, possibly combining it with the service for *Spooler* messages (*Spooler_Service*) to direct all *GDMA* messages to one service instead of two.
- **Poller_Status=On/Off** - This controls whether the poller generates the status messages at all. If you turn it off, that host will not generate these messages. You should remove the `gdma_poller` service from the host in this case.
- **Spooler_Status** - This controls whether the spooler sends status messages, and, if so, when. If it's off, no spooler messages will be sent. If it is on, you will get a message each time the spooler runs, which is every 30 seconds by default. If you set it to Updates, it will send you a message only if there has been an error, or something has changed (such as a recovery from an error). *GDMA* sets this to *Updates* by default.
- **Warning_Threshold, Critical_Threshold=(integer percent)** - Applied to the poller, these control when you will get notified that there is a problem with the poller chewing up more than the indicated percentage of its cycle time to perform a single check. That situation is something you probably want to know about, since it means you may be trying to check too frequently, or to check too many services. Remember, *GDMA* is designed to use very little CPU resources, so it has some built-in waits, and (except in multi-host mode), it actually does not perform checks in parallel. If you are getting close to the limit and you can manage it, you can change these thresholds from their defaults (60 and 80 percent) to cut down on false positives for these services.

Using fully qualified host names

If you manage multiple data centers, or if you have domains of systems that you want to use GDMA on, and want those systems to report all to the same GroundWork server, you may run into a situation where the same short host name is used for more than one host. GroundWork Monitor uses Nagios, which requires that host names be unique. For GDMA, this presents a problem, as the host name is automatically determined on the GDMA system, so it is possible to have two systems reporting status, and the GroundWork server will only represent them as one.

As long as the domain names differ, however, this can be accommodated by setting up the GDMA hosts in GroundWork Monitor using fully qualified names. Thus a host like `server1.foo.com` can be distinguished from `server1.bar.com`, and GroundWork Monitor will be able to tell them apart.

To enable this feature, use the parameter `Use_Long_Hostname`. Set this parameter to `"on"` in the `gdma_auto.conf` file on the GDMA client, and change the host name in GroundWork Monitor Configuration to be the fully qualified name. This will change the way GDMA reports its results for services, and though the first run with the new settings will generate a single spooler messages under the old (short) hostname, subsequent cycles will have the long fully-qualified name used for poller and spooler messages as well.

Forcing determination of GDMA client hostname

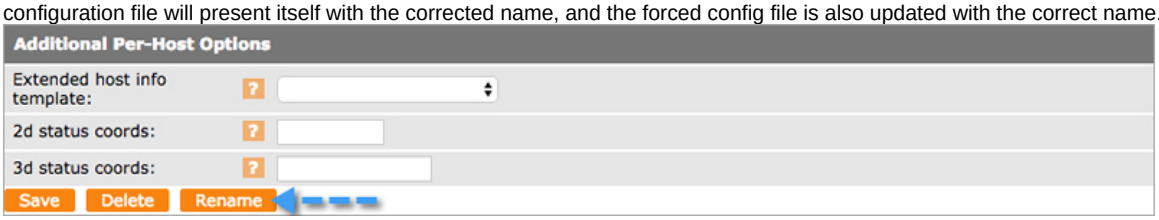
The `Forced_Hostname` directive is used to force determination of the GDMA client hostname to a fixed value. This option is fully supported in the latest GDMA release, on all platforms. `Forced_Hostname` is an optional directive in the GDMA client config files. The value is the exact hostname (unqualified or fully-qualified, as specified) to be used in place of any dynamic determination of the hostname. GDMA will lowercase the supplied value before use, to correspond to the uniform use of lowercase hostnames in the rest of GDMA.

Additionally, this option is used in support of GDMA auto-registration so that when a GDMA client successfully registers itself with the GroundWork server, the hostname that the server determined was correct for this client gets returned to the client. The client then stashes that hostname as the value of the `Forced_Hostname` option in the new `gdma/config/gdma_override.conf` file, so both the poller (which received the hostname from the server) and the spooler (which passively accepts this instruction) know the proper hostname to use even if those components are bounced. This way, the base `gdma_auto.conf` configuration file is never modified by the auto-registration processing. When the `Forced_Hostname` is used in this manner for auto-registration, any value, which may have been manually set in the `gdma/config/gdma_auto.conf` file, will be ignored in favor of the value in the `gdma/config/gdma_override.conf` file.

Also note that the GDMA auto-registration feature provides special support on the server for forcing the assignment of hostnames to particular values for certain client hosts, via the `<hardcoded_hostnames>` section of the `config/register_agent.properties` file.

> ⚠ **KNOWN ISSUE - Renaming a GDMA Host**

- Known Issue:
  Using the Rename button, located at the bottom of the host detail screen (Configuration > Hosts > Hosts > {hostgroup} > {host} > Detail), to rename a GDMA host will cause the host to stop receiving checks. This action fails to overwrite the agent configuration on the remote machine and restart the remote agent to allow it to pull the new configuration. Once the workaround below has been completed, a new configuration file will present itself with the corrected name, and the forced config file is also updated with the correct name.



- Workaround:
  This must happen on the GDMA client only after the next Commit after the server-side Rename occurs. That way, the new setup will be in place on the server when the updated client reaches for it. Also note that, after the Commit, the administrator should re-build externals, so make sure they are all up-to-date, before making changes on the GDMA client.
  - Stop GDMA service
  - Delete the forced config and the host named config files
  - Start GDMA service
- GDMA client files to delete:
  - `gdma/config/gdma_override.conf` (which file is supplied to the GDMA client by the GroundWork server during auto-registration, and holds a `Forced_Hostname` directive specifying the form and content of the hostname that the client is to use when interacting with the server)
  - `gdma/config/gwmon_`*`hostname`*`.cfg` (which file, named using the precise form of the client hostname which the client should be using, is supplied to the GDMA client by the GroundWork server, and contains host and service externals from the server which are relevant to this client, note this is the actual hostname of the GDMA client, not the literal characters "hostname")

Checking multiple hosts from a single Linux GDMA host

This is an advanced feature that is useful when checking clustered hosts, or simply setting up some lightweight monitoring of nearby hosts from a single Linux GDMA.

- The Linux GDMA must be installed using the command-line option `--multihost 1`
- You need to set up a configuration group for all hosts that will be checked in this way, with a specific build directory for the config files
- You need to create that build directory, and set permissions

2.0 Downloading New Plugins Automatically

GDMA (2.2.1 and later) with GroundWork Monitor (6.4 and later) allows you to download new plugins (or new versions of old plugins) automatically to specific versions of the GDMA. The following instructions illustrate how to enable this feature.

> ⚠ This feature is disabled by default.

Enable the plugin download feature

1. Edit the file:

```
/usr/local/groundwork/config/foundation.properties
```

change:

```
gdma.plugin.upload.enable=false
```

to:

```
gdma.plugin.upload.enable=true
```

and save the file.

2. Edit the file:

```
/usr/local/groundwork/apache2/conf/httpd.conf
```

change:

```
#Alias /plugin_download "/usr/local/groundwork/apache2/htdocs/agents/plugin_download"
```

to:

```
Alias /plugin_download "/usr/local/groundwork/apache2/htdocs/agents/plugin_download"
```

and save the file.

3. Restart Foundation, run as `root`:

```
/etc/init.d/groundwork restart gwservices
```

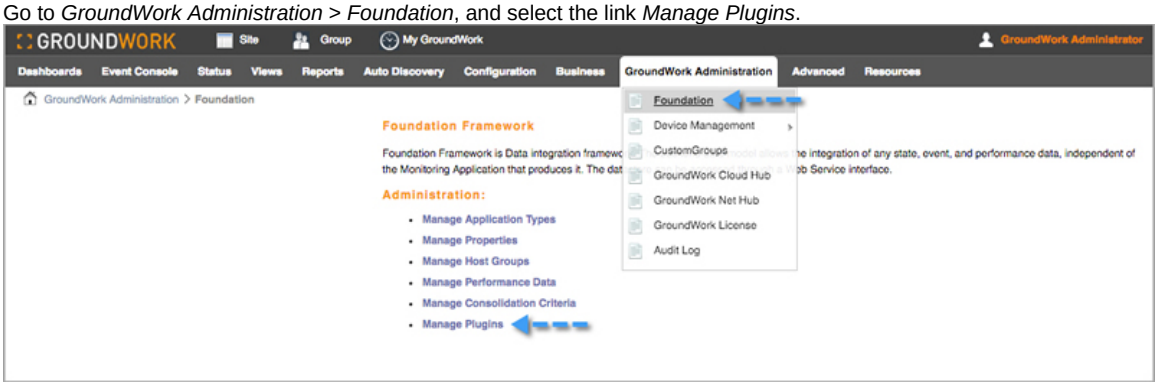
4. Restart Apache, run as `root`:

```
/etc/init.d/groundwork restart apache
```

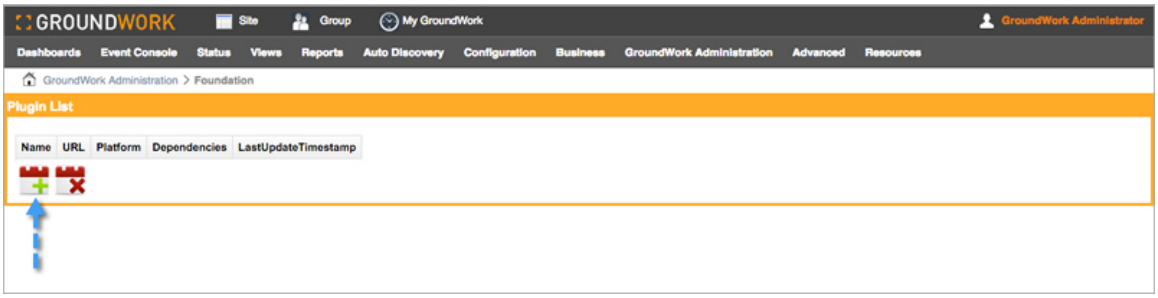

This will log off all users.

Upload new plugin files

1. Go to *GroundWork Administration* > *Foundation*, and select the link *Manage Plugins*.

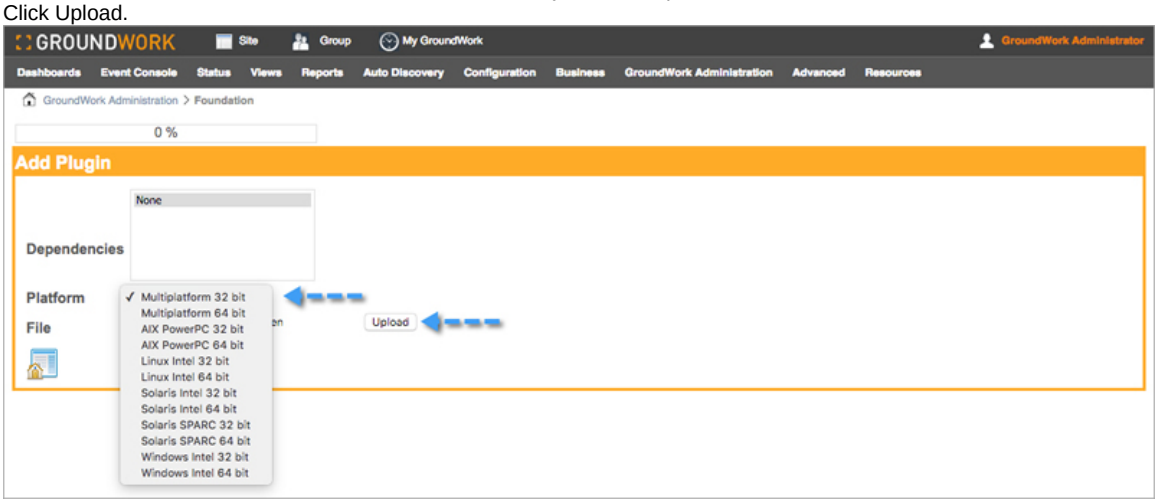


2. Click the add icon from the Plugin List screen.

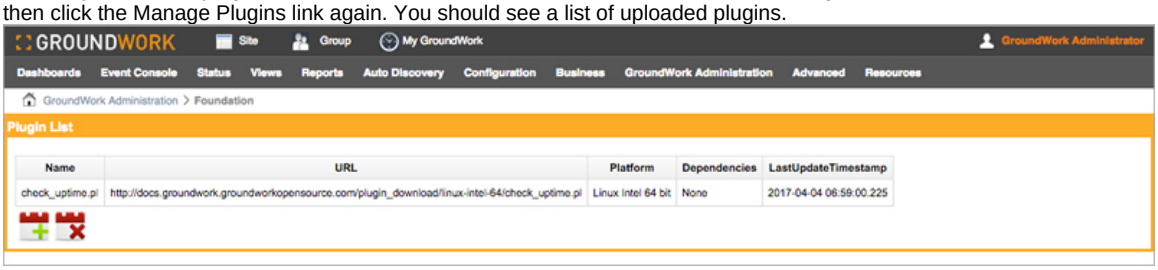


3. Choose the platform and architecture the plugin being uploaded will be used from the drop-down. If your plugin will work on all the platforms on which you run *GDMA*, choose one of the Multiplatform options. If you want to run the plugin on both 32-bit and 64-bit platforms, you will need to upload it twice, once for Multiplatform 32 bit and once for Multiplatform 64 bit. Generally, however, you will have to upload a different version of a plugin for each platform you want to use it on.

   The names of the plugins for different operating systems or bit widths can be the same, though you will need to keep them in separate, well-defined places on your source machine to keep track of which copy is for which platform. The uploaded plugins will be stored in separate directories on the server, so there will be no confusion there. For instance, a plugin written as a shell script might work on just Linux but for both 32 bit and 64 bit versions of that OS, so you would upload it once for *Linux Intel 32 bit* and once for *Linux Intel 64 bit*. Click Upload.



4. To view your current plugins, click the *Home* icon in the lower-left corner of the Add Plugin box. Or click Foundation in the menu, and then click the Manage Plugins link again. You should see a list of uploaded plugins.



5. In GroundWork Monitor (6.5 and later), if you need to update a plugin (keeping the same name, but changing its content), you will need to delete and re-add it on each platform. The extra step involved in a delete/add as opposed to just re-uploading will be addressed in a future release.

Tell GDMA to download the plugins

1. Go to *Configuration* > *Hosts*, *Host Externals* > *Modify* and select a host external for the hosts that will be downloading plugins. Alternatively you can enable this for individual hosts by opening the host external for the specific host you want, but changing the host external for all like hosts will make it the downloads apply to many hosts at once. You may wish to try this on one host, and make sure you have it right before rolling out to all hosts of a given type.

   ⚠ Only modify host externals for hosts running *GDMA* (2.2.1 or later). Older versions will error out if these parameters are placed in the host externals.

   - Add the following lines to the selected host external where <Server Name> is the resolvable name of the GroundWork server from the point of view of the GDMA:

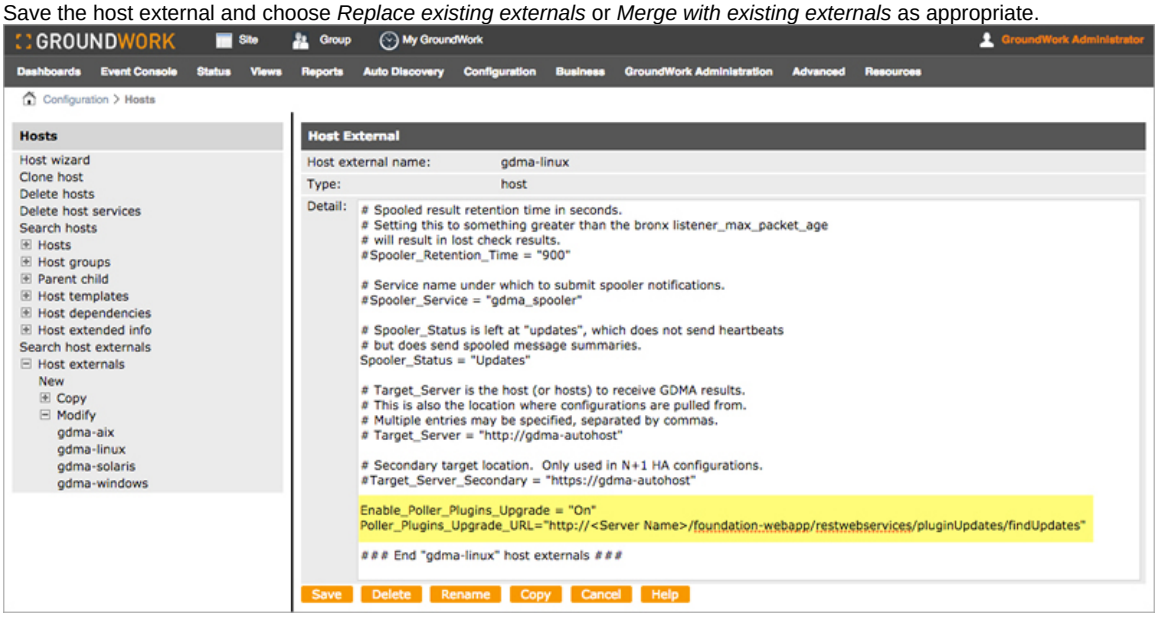
```
Enable_Poller_Plugins_Upgrade = "On"
Poller_Plugins_Upgrade_URL="http://<Server
Name>/foundation-webapp/restwebservices/pluginUpdates/findUpdates"
```

You may also use an https URL if you have configured *GroundWork Monitor* to use *HTTPS*, e.g.,:

```
Poller_Plugins_Upgrade_URL="https://<Server
Name>/foundation-webapp/restwebservices/pluginUpdates/findUpdates"
```

Important Note
When using HTTPS, the <Server Name> must exactly match what is in the server's SSL certificate (typically, a fully-qualified name). For more information regarding SSL support see the Bookshelf document How to enable HTTPS.

- Save the host external and choose *Replace existing externals* or *Merge with existing externals* as appropriate.



2. And build externals, *Configuration* > *Control* > *Build Externals*. Plugins will be downloaded to the GDMA `libexec` directory, and will be flagged executable by the user running the GDMA process (*gdma* by default). Existing plugins with the same file name will be replaced.

Plugin dependencies (UNIX versions)

You may want to use a plugin that depends on a library that GroundWork Monitor does not supply with the GDMA. For Windows systems, this is not usually the case, since though one could characterize a .dll file as a dependency, most Windows plugins are either VBScript (executed by `cscript.exe`), PowerShell (executed by `powershell.exe`) or Windows `.bat` batch files (executed by `cmd.exe`). If you need to download a .dll file to Windows, you should simply add it as you would a plugin, and it will be copied to the `libexec` subdirectory of the GDMA, where it will be available for the plugin.

If you are using the UNIX GDMA (Linux, Solaris or AIX), you may have access to the library you need. If you have this library (typically, one or more `.so` files) available, you can transfer it to the GDMA host into the `groundwork/common/lib` directory. Simply perform the above procedure, uploading the dependency files first, choosing platform and architecture as you would for the plugin. Then, when you have the library (or libraries) uploaded, upload the plugin, selecting the libraries in the dependency screen.

In this example, the `test001.so` library is needed by the new `check_my_app` plugin. Selecting the dependency when the plugin is loaded identifies `test001.so` as a dependency, which goes to the `groundwork/common/lib` directory, and `check_my_app` as a plugin, which gets placed in the `groundwork/nagios/libexec` directory.

If you update an existing plugin, the new version will simply overwrite the old. No special action will be taken to preserve the original plugin.

Plugins are downloaded only as needed. The system will check to see if the plugins are new, and have a different ND5 sum prior to downloading, so there is no downside to keeping a large number of plugins and dependencies on the GroundWork server.

You may remove a plugin or dependency (or several at once) at any time from the *Manage Plugins* screen, by selecting the files to be removed and clicking the *X* button.

3.0 Working with PowerShell

The GDMA for Windows can be used to run Windows PowerShell Commandlets, or small PowerShell programs that you write or modify to return results that can be interpreted as status and performance data by GroundWork Monitor. Especially in 64-bit environments, this is quite a powerful way to monitor your Windows systems.

GroundWork Monitor (6.4 and later) and GDMA (2.2.1 and later) come with some sample PowerShell scripts that leverage commandlets to check some otherwise difficult-to-access data about 64-bit Windows systems. This section will take you through getting GDMA set up to run these examples.

Installing PowerShell

Before you can run the PowerShell plugins, you must have PowerShell installed and working on your Windows system. Please see http://technet.microsoft.com/en-us/library/ee692944.aspx for information on getting PowerShell running.

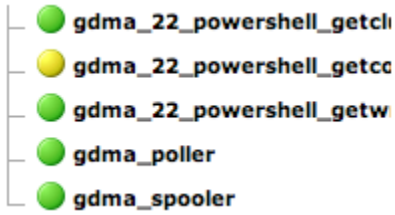
In particular you MUST enable PowerShell to run scripts. To do this, launch each version of the PowerShell interpreter on your system (both the 64 bit Windows PowerShell and the 32-bit Windows PowerShell (x86)), and type:

```
Set-ExecutionPolicy RemoteSigned
```

Loading Example Profile

You will notice a new GDMA profile; the host profile gdma-22-windows-host.xml contains the service profile gdma-22-powershell, which in turn contains three services:

- **gdma_22_powershell_getcluster**: A plugin to get the status of a Microsoft cluster server, if it is installed
- **gdma_22_powershell_getcounter**: A generic plugin that gets any perf counter, but must be modified before it will be useful
- **gdma_22_powershell_getwmi**: An example of calling a WMI counter (CPU in this case) from a PowerShell script
  The modification of these example scripts in to more useful or specific uses is ongoing at GroundWork, and will doubtless be the source for many more profiles. For the moment, however, these are a starting point. To get these examples loaded:
  1. Go to *Configuration* > *Profiles* > *Profile Importer*.
  2. Select *gdma-22-windows-host.xml* from the list.
  3. Click *Import*. This will result in a host profile, which you can then apply to a host. Make this the host on which you have installed and enabled PowerShell previously. Once you commit, and build externals these services will be active. Of course, it's normal not to have OK status on an un-configured service, so `gdma_22_powershell_getcounter` will be in warning until you modify the plugin and give it something to monitor. Also, unless you have Microsoft Cluster Server loaded on the host, you will not get logical output from `gdma_22_powershell_getcluster`.



> ⚠ The service externals are not generic, and may need to be modified to work with your system.

The service externals assume the default location of PowerShell and GDMA. The example here explicitly calls

`powershell.exe` in the default location, and passes it the full, no-spaces version of the path to the PowerShell script to run. The quoting used is standard, and can be used in most cases. See 'Command='Line Tricks below.

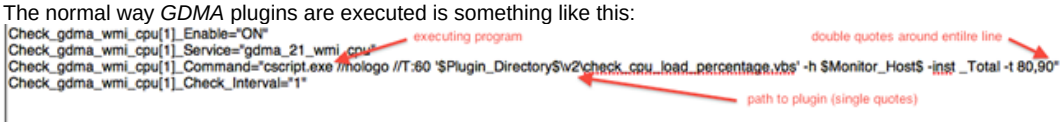
> ⚠ If you instal GDMA in a different directory, you will need to modify this path. The short form Progra~2 expands out to Program Files (x86), which is where GDMA, a 32-bit program, is installed in 64-bit Windows by default.

```
Check_gdma_powershell_getcounter[1]_Command="c:\windows\system32\WindowsPowerShell\v1.0\powe
'C:\Progra~2\groundwork\gdma\libexec\v3\getCounter.ps1'"
```

4.0 'Command='line Tricks

In some cases you may want to execute a check command that does not fall under the usual formatting rules. In particular if you want to make your own plugins, or integrate a plugin that you download off of the Internet, you may need to adjust the format of the command line in the *Service External* you use to control the plugin execution.

The normal way *GDMA* plugins are executed is something like this:



The entire command line is enclosed in double quotes. Within this enclosure, there are two main sections. The first is the executing program (`cscript.exe`, in this case, but in the PowerShell example in the previous section, it was `powershell.exe`). Any arguments to the executing program come next. If you are using UNIX, the executing program is implicit: the command shell. In that case you can stop there, for example:

```
Check_gdma_linux_mem[1]_Command="check_mem.pl -F -w 20 -c 10"
```

In Windows, as well, you can run compiled programs as plugins in this way without any further specifications. Just be sure to include the enclosing double quotes.

The second main part is the interpreted plugin, with its full path, encased in single quotes. You can usually use the `$Plugin_Directory$` macro, and this will be replaced with the location of the plugins in your GDMA installation. Note that you can have subdirectories off of this main location, for example many of the vbscript plugins we supply are stored in the v2 subdirectory. PowerShell plugins are stored in v3 by convention.

After the single-quoted full path to the interpreted plugin, and before the terminating double quote mark, you can supply any arguments to the interpreted plugin. The GDMA will understand bare arguments, arguments specified with a dash "-", or with a slash "/".

5.0 Process for Macro Substitutions

GDMA macro substitution is done in two places:

1. On the server, when externals are built
2. On the client, when checks are executed

Here are all the variables that are substituted into the `host-config` file content, both when externals are built on the server, and when the contents of that file are interpreted on the client.

1. When externals are built on the server:
    a. Service external macros are substituted first. For each host, all the service externals for that host are processed first and all of their service-related macro substitutions are done at that time. Listed here are macro references that may be used in service externals:
        - `$SERVICEDESC$` expands to the full name of the service, as you would expect, including the appropriate instance name suffix in an instance-level external. This should be consistent with the use of this macro name throughout the rest of the system.
        - `$BASESERVICEDESC$` expands to the name of the service. For a service without instances, this will be the same value as `$SERVICEDESC$`. For a service with instances, this will be the name of the base service, without any instance name suffix appended.
        - `$INSTANCE$` expands to a simple integer which counts the number of copies of a given external as possibly multiple copies are created for service instances. This macro will generally be used within the square brackets of service-externals lines. For a base service without any service instances defined, this number will be a constant `1`.
        - `$INSTANCESUFFIX$` expands to the instance name suffix as defined in Monarch, but with a leading underscore (if any) elided. This makes it easier to customize the service externals definitions with a string reflecting just the critical part of the service instance name suffix, while keeping an underscore in the instance name suffix for overall readability of the full service name.

- $ARG#$ macros ($ARG1$, $ARG2$, etc.) expand to values defined in the Monarch UI at either the generic-service, host-service, or service-instance level, depending on the complexity of your setup and what inheritance options you have chosen.
- Attempting to use these macros to best effect and to generalize service externals definitions as much as possible, the following general form is recommended: (This example is not trying to establish any convention for the ordering or use of the $ARG#$ macros. Instead, it is illustrating the intended uses of the $BASESERVICEDESC$, $INSTANCE$, $SERVICEDESC$, and $INSTANCESUFFIX$ macros.)

```
Check_$BASESERVICEDESC$[$INSTANCE$]_Enable="ON"
Check_$BASESERVICEDESC$[$INSTANCE$]_Service="$SERVICEDESC$"
Check_$BASESERVICEDESC$[$INSTANCE$]_Command="check_foo -w $ARG1$ -c $ARG2$ -i
$INSTANCESUFFIX$"
Check_$BASESERVICEDESC$[$INSTANCE$]_Check_Interval="$ARG3$"
```

IMPORTANT UPDATE

What we have above, as the recommended boilerplate construction for service externals, will work cleanly for all setup cases for GDMA **2.7.0 and later**. It will also work adequately when generating externals for earlier GDMA releases, for services with a `_Check_Interval` of "1". **However, if you have earlier GDMA releases deployed**, it is necessary to switch to an alternate construction if you have set the `_Check_Interval` in the service external to a value greater than 1 (in this example boilerplate, we assume that $ARG3$ might expand to a value other than 1) and you have multiple instances for the service:

The alternate boilerplate form would look like:

```
Check_$SERVICEDESC$[1]_Enable="ON"
Check_$SERVICEDESC$[1]_Service="$SERVICEDESC$"
Check_$SERVICEDESC$[1]_Command="check_foo -w $ARG1$ -c $ARG2$ -i $INSTANCESUFFIX$"
Check_$SERVICEDESC$[1]_Check_Interval="2"
```

The expanded forms could be these, assuming instance names of `_first` and `_second` for the instances of the `foo` service on this host in Monarch:

```
Check_foo_first[1]_Enable="ON"
Check_foo_first[1]_Service="foo_first"
Check_foo_first[1]_Command="check_foo -w 10 -c 20 -i first"
Check_foo_first[1]_Check_Interval="2"

Check_foo_second[1]_Enable="ON"
Check_foo_second[1]_Service="foo_second"
Check_foo_second[1]_Command="check_foo -w 15 -c 25 -i second"
Check_foo_second[1]_Check_Interval="2"
```

Specifically, $BASESERVICEDESC$ gets manually changed to $SERVICEDESC$, and [$INSTANCES$] gets manually changed to [1]. GDMA itself is in this case unaware that these are multiple service instances; it just treats them as completely separate services, each with only one instance.

b. Then, Nagios resource macros are substituted. These are $USERn$ references, as defined under *Configuration* > *Control* > *Nagios resource macros*.
c. Next, Monarch group macros are substituted, as defined under *Configuration* > *Groups* > *Macros* (to first establish the existence of a given macro name) and then under *Configuration* > *Groups* > *Groups* > *<group name>* > *Detail* > *Macros*.
   - A given host can be a member of one or more Monarch groups. This membership can be either via assignment of the host to a specific Monarch group, or indirect via assignment of the host to a hostgroup, and assignment of the hostgroup to a Monarch group.
   - Typically, when a host is a member of more than one Monarch group, these Monarch groups are arranged in some kind of parent-group/sub-group tree (which can be quite complex). If the host is a member of multiple Monarch groups like that, then when the externals file for the top-parent Monarch group is created, group-macro substitutions are made in order from sub-groups to parent groups, up the chain of ancestry. This allows the sub-group macro definitions to override parent-group macro definitions, which is an important use case for having sub-groups.
   - When externals are built for a given host, they are really only built for each top-level Monarch group associated with that host; you don't get an extra externals file for each sub-group to which the host belongs.
d. Finally, a few fixed-name macros are substituted, in order: $HOSTNAME$, $HOSTADDRESS$, and $HOSTALIAS$. These values are drawn from the Monarch configuration for this host.

2. When externals are interpreted on the client:
   a. $Plugin_Directory$ (drawn from the client's configured `Poller_Plugin_Directory`) and $Monitor_Host$ (the name of the host for which the check is being run) are substituted first into each plugin execution string.

b. Option parameters are substituted into the plugin execution string, in essentially arbitrary order (you can't depend on a specific ordering of these substitutions).

This last category, of option parameters, seems not to be documented anywhere, rather to our chagrin. We'll get this corrected; this capability ought to be listed in the GDMA Configuration Reference page, under Per Service Configuration Parameters, along with some realistic examples of its use. In the meantime, here are the details, drawn from reverse-engineering the code.

In addition to the documented parameters:

Check_{service}_Enable
Check_{service}_Service
Check_{service}_Command
Check_{service}_Timeout
Check_{service}_Check_Interval

which are used in service externals like these lines:

Check_gdma_linux_swap[1]_Enable="ON"
Check_gdma_linux_swap[1]_Service="linux_swap"
Check_gdma_linux_swap[1]_Command="check_swap -w 10% -c 5%"
Check_gdma_linux_swap[1]_Check_Interval="1"

you can also define any number of "Parm" parameters for a given service, in any of several forms:

Check_Disk[1]_Parm_--warning = "10%"
The value of a double-quoted parameter value is set to the string that is enclosed in quotes.

Check_Response_Servlet[1]_Parm_-n = Monitor_Server[1]
Check_This_Servlet[1]_Parm_-n = Check_Other_Servlet[1]_Parm_-n
If there are no quotes after the "=" character, the value of this parameter can be set to the value of another parameter that has already been defined. (Cross-references like this can have considerable complexity, which we are not describing in detail here.)

Check_Response_Servlet[1]_Parm_-n = Monitor_Server[1],Monitor_Server[2]
In the case just described, comma-separated multiples like this are supported.

Check_Response_Servlet[1]_Parm_-n = gw.company.com
If there are no quotes around the value, and that value does not represent some other parameter which is already defined, the value will be taken as-is.

Check_Response_Servlet[1]_Parm_-n = gw1.company.com,gw2.company.com
In the case just described, comma-separated multiples like this are supported.

Check_Disk[1]_Parm_--errors-only
If there is no "=" character in the parameter line, the parameter value is set to an empty string.

These parameters will be substituted into the plugin execution string as follows:

1. A parameter with a name like "Parm_--ABC" will be appended to the command line as " --ABC=$value", where $value is the value of that parameter as listed above. Note that in the present GDMA code, $value will not itself be surrounded by additional quotes when it is used in the command line this way, which can be problematic (any spaces or shell metacharacters in the $value will not be protected from interpretation by the shell). If you need such protection, the $value itself should contain single-quote characters to provide such protection:

   Check_Load[1]_Parm_--message = "'A protected string.'"

2. A parameter with a name like "Parm_-DEF" will be appended to the command line as "-DEF $value", where $value is the value of that parameter as listed above. The same issue with quoting applies here.

   It looks like there is some inconsistency in the way the present code works. You would probably want this specification:

   Check_Disk[1]_Parm_--errors-only

   to be substituted into the plugin execution string by having the string `--errors-only` be appended to the command line. But it looks like the present code will instead append `--errors-only=`; this can be considered a bug. There are probably some other boundary cases that are also not handled as well as they should be. We will need to clean up these things in a future GDMA release.