

This file stays here for reference, its contents, updated, have been moved to <https://github.com/stdmod/>

Please use GitHub for comments, proposals, samples, pull requests.

stdmod account on GitHub is configured as an organization, if you want to actively participate to the definition of a common naming standard (and eventually a common modules set) ask for commit rights.

Puppet Modules Naming Standards - Draft 0.0.1

Proposals for naming conventions for the parameters used in classes and defines of a module.

A “standard” module MUST NOT require all of the proposed naming conventions, but SHOULD, for the parameters supported, keep these standards.

Consider this document as a temporary place where to spark the discussion.

Usage patterns

<pre>class { 'my': template => 'module/path/file.erb', }</pre>	<pre>hieradata ('my::template') { "my::template" : modules/path/file/.erb, }</pre>
---	--

with Puppet 3 data bindings an automatic Hiera lookup is done for each parameter of a class

Names and Namespaces layouts

Class parameters

Some possible patterns:

- 1- Place a common suffix for any kind of parameter
- 2- Place suffixes for some specific params and leave more general names of general ones

Enhancement for case 1	Examples for case 1	Examples for case 2
ensure audit noop package package_version	ensure package_name package_version service_name service_ensure file_path file_source	ensure package version service service_ensure file_path source

service service_ensure service_enable service_notify ? file file_source file_template file_options_hash file_owner file_group file_mode other_file other_file_source other_file_template other_file_options_hash ... dir dir_source dir_recurse dir_purge class_my class_dependency resource_hash install_mode install_source install_destination install_precommand install_postcommand monitor monitor_tool monitor_target monitor_port monitor_url monitor_process monitor_service monitor_config_hash	file_template dir_path dir_source	template dir_path dir_source
--	---	------------------------------------

firewall firewall_src firewall_dst		
--	--	--

Defines parameters

Parameters related to files managed by the define

ensure file file_template file_options_hash		ensure file template options
--	--	---------------------------------------

Parameters related to commands executed by the define

ensure exec_environment exec_creates exec_refreshonly exec_options_hash		ensure environment creates refreshonly options
---	--	--

Manage removal of the resources provided by the module:

```
# [*ensure*]
# String. Default: present
# Manages package installation and class resources. Possible values:
# * 'present' - Install package, ensure files are present.
# * 'absent' - Stop service and remove package and managed files
#
```

Manage Package(s)

```
# [*package*]
# String. Default: Defined in stdmod::params.
# Name of the package to install. Can be an array.
#
# [*version*]
# String. Default: undef
```

If set, the defined package version is installed. Possible values:
* 'x.x.x' - The version to install
* 'latest' - Ensure you have the latest available version.

Manage Service(s)

[*service*]
String. Default: Defined in stdmod::params.
Name of the service to manage.

[*service_ensure*]
String. Default: running
Define the current service status. The same values as service#ensure apply.

[*service_enable*]
String. Default: true
Define the service status at boot time. The same values as service#enable apply.

[*service_notify*]
Boolean. Default: true
Define if changes to configurations automatically trigger a restart
of the relevant service(s)
#

Where needed, parameters that manage the service initialization scripts

[*initialize_template*]
Template file to use to populate the initialization script

[*initialize_path*]
Path of the initialization script (ie: /etc/init.d/stdmod)

[*run_options*]
The explicit options to pass to the starting command.
This value is appended to existing -f and -l parameters
#

Manage the [main] configuration file(s)

[*file_path*]

```

# String. Default: Defined in stdmod::params.
# Path of the [main] configuration file.

# [*file_source*]
# String or Array. Default: undef. Alternative to 'template'.
# Sets the content of source parameter for main configuration file.
# If defined, stdmod main config file will have the param: source => $source
# Example: source => 'puppet:///modules/site/stdmod/stdmod.conf',
#
# [*file_template*]
# String. Default: undef. Alternative to 'source'.
# Sets the path to the template to use as content for main configuration file
# If defined, stdmod main config file has: content => content("$template")
# Example: template => 'site/stdmod.conf.erb',
#
# [*file_options*]
# Hash. Default undef. Needs: 'template'.
# An hash of custom options to be used in templates to manage any key pairs of
# arbitrary settings. Usually this is implemented to map directly to configuration statements.
# Special values MUST be documented in the module.
#

```

Manage the whole configuration directory

This complete section is optional depending on the managed software having an configuration directory.

```

# [*dir_path*]
# String. Default: Defined in stdmod::params.
# Path of the configuration dir.
#
# [*dir_source*]
# String or Array. Default: undef
# If set, the main configuration dir is managed and its contents retrieved
# from the specified source.
# Example: dir_source => 'puppet:///modules/site/stdmod/conf.d/',
#
# [*dir_recurse*]
# String. Default: true. Needs 'dir_source'.
# Sets recurse parameter on the main configuration directory, if managed.
# Possible values:
# * 'true|int' - Regular recursion on both remote and local dir structure

```

```
# * 'remote' - Descends recursively into the remote dir but not the local dir
# * 'false' - No recursion
#
# [*dir_purge*]
# Boolean. Default: false
# If set to true it removes all the files on the configuration directory that are not
# deployed via Puppet (with params like file_source, file_template and dir_source)
# Set it to true when you want to be sure that Puppet manages completely
# the configuration directory.
```

Manage any extra configuration file, service or package with a common prefix (here 'otherfile')

```
# [*other_file_path*]
# String. Default: Defined in stdmod::params.
# Path of the [main] configuration file.

# [*other_file_source*]
# String or Array. Default: undef. Alternative to 'template'.
# Sets the content of source parameter for otherfile configuration file.
# If defined, otherfile.conf file will have the param: source => $source
# Example: source => 'puppet:///modules/site/stdmod/stdmod.conf',
#
# [*other_file_template*]
# String. Default: undef. Alternative to 'source'.
# Sets the path to the template to use as content for main configuration file
# If defined, stdmod main config file has: content => content("$template")
# Example: template => 'site/stdmod.conf.erb',
#
# [*other_file_options*]
# Hash. Default undef. Needs: 'template'.
# An hash of custom options to be used in templates to manage any key pairs of
# arbitrary settings.
#
```

Manage interoperability with 3rd party modules resources

```
# [*dependency_class*]
# String. Default: stdmod::dependency
```

```
# Name of a class that contains resources needed by this module but provided
# by external modules. You may leave the default value to keep the
# default dependencies as declared in stdmod/manifests/dependency.pp
# or define a custom class name where the same resources are provided
# with custom ways or via other modules.
# Set to undef to not include any dependency class.
#
```

Add extra custom resources related to the class or define

```
# [*my_class*]
# String. Default undef.
# Name of a custom class to autoload to manage module's customizations
# If defined, stdmod class will automatically "include $my_class"
# Example: my_class => 'site::my_stdmod',
#
# [*resources_type*]
# String. Default file.
# The type of resources to create with create_resources
#
# [*resources_hash*]
# Hash. Default undef.
# An hash to pass to the create_resources function.
#
# The above resource_ parameters as used with
#   create_resources($resources_type, $resources_hash )
#
#
# ALTERNATIVE?
# [*create_resource_hash*]
# Hash. Default file.
# A single hash that contains the complete list of resources to create (with both parameters
and types of resources to create)
# The module has to manage and split the hash to feed the create_resources function.
```

Manage auditing and noop

```
#
# [*audits*]
```

```
# String or array. Default: undef.
# Applies audit metaparameter to managed files.
# Ref: http://docs.puppetlabs.com/references/latest/metaparameter.html#audit
# Possible values:
# * '<attribute>' - A name of the attribute to audit.
# * [ '<attr1>' , '<attr2>' ] - An array of attributes.
# * 'all' - Audit all the attributes.
#
# [*noops*]
# Boolean. Default: false.
# Set noop metaparameter to true for all the resources managed by the module.
# If true no real change is done is done by the module on the system.
#
```

Manage alternative installation options

```
# [*install_mode*]
# Kind of installation to attempt:
#   - package : Installs stdmod using the OS common packages
#   - upstream : Installs (compiled) stdmod using the upstream source
#
# [*install_source*]
# URL to retrieve the upstream package.
# Used if install => "upstream"
# Default is from upstream developer site. Update the version when needed.
#
# [*install_destination*]
# Path where source package is exploded.
# Used if install => "upstream"
#
# [*install_precommand*]
# Optional custom command to execute before installing the source tarball/zip.
# Used if install => "upstream"
# Check stdmod/manifests/params.pp before overriding the default settings
#
# [*install_postcommand*]
# A custom command to execute after installing the source tarball/zip.
```

```
# Used if install => "upstream"
# Check stdmod/manifests/params.pp before overriding the default settings
#
```

Manage monitoring

```
# [*monitor*]
# Set to 'true' to enable monitoring of the services provided by the module
# Can be defined also by the (top scope) variables $graylog2_monitor
# and $monitor
#
# [*monitor_tool*]
# Define which tool(s) to use for monitoring
#
# [*monitor_target*]
# The Ip address or hostname to use as a target for monitoring tools.
# Default is the fact $ipaddress

# [*monitor_port*]

# [*monitor_url*]

# [*monitor_process*]

# [*monitor_service*]

# [*monitor_config_hash*]
```

Manage firewalling

```
#
# [*firewall*]
# Set to 'true' to enable firewalling of the services provided by the module
#
# [*firewall_src*]
# Define which source ip/net allow for firewalling. Default: 0.0.0.0/0
#
# [*firewall_dst*]
# Define which destination ip to use for firewalling. Default: $ipaddress
```

Parameters for defines that create files

```
# [*ensure*]  
# String. Default: present  
# Manages configfile presence. Possible values:  
# * 'present' - Install package, ensure files are present.  
# * 'absent' - Stop service and remove package and managed files  
#  
# [*template*]  
# String. Default: Provided by the module.  
# Sets the path of a custom template to use as content of configfile  
# If defined, configfile file has: content => content("$template")  
# Example: template => 'site/configfile.conf.erb',  
#  
# [*options*]  
# Hash. Default undef. Needs: 'template'.  
# An hash of custom options to be used in templates to manage any key pairs of  
# arbitrary settings.  
#  
define stdmod::configfile (  
  $ensure = present,  
  $template = 'stdmod/configfile.erb' ,  
  $options = " ,  
  $ensure = present ) {
```

Parameters for defines that execute commands

```
# [*refreshonly*]  
# Boolean. Optional. Default: true  
# Defines the logic of execution of the script when Puppet runs.  
# Maps to the omonymous Exec type argument.  
#  
# [*creates*]  
# String. Optional. Default: undef  
# Defines the logic of execution of the script when Puppet runs.  
# Maps to the omonymous Exec type argument.  
#  
# [*onlyif*]  
# String. Optional. Default: undef  
# Defines the logic of execution of the script when Puppet runs.  
# Maps to the omonymous Exec type argument.
```

```
#
# [*unless*]
# String. Optional. Default: undef
# Defines the logic of execution of the script when Puppet runs.
# Maps to the omonymous Exec type argument.

# [*exec_environment*]
# Define any additional environment variables to be used with the
# exec commands. Note that if you use this to set PATH, it will
# override the path attribute. Multiple environment variables
# should be specified as an array.

# [*path*]
# Define any additional environment variables to be used with the
# exec commands. Note that if you use this to set PATH, it will
# override the path attribute. Multiple environment variables
# should be specified as an array.
```

Generic parameter to manage users creation, where a class or defines needs to create a user

```
# [*user_create*]
# Set to true if you want the module to create the process user of stdmod
# (as defined in $logstagh::process_user). Default: true
# Note: User is not created when $stdmod::install is package
#
```

Parameters for classes or defines that manage Java applications.

```
#
# [*java_opts*]
# String. Default: Defined in stdmod::params.
# Optional string to add to JAVA_OPTS
# [...]
```

Have a minimal naming convention for for apache::vhost / nginx::vhost ...

Params like \$ensure, \$template, \$options (?) to manage the file created.

For settings used in the template (DocumentRoot, Aliases ...) many possible options:

Define a brand new naming standard?

Adapt to puppetlabs-apache? (standard de facto?)

Define an higher level module that manage virtualhosts for different web servers?? (ie: webserver::vhost or web::vhost or vhost)

Parameters for database operations

\$db_name - The database to use
\$db_user - User to grant the permissions to.
\$db_password - Password for the user.
\$db_privileges - Privileges to grant to the user.
\$db_host - Database host
\$db_port - Database port

Parameters for mysql::grant? database::grant?

Higher level modules

Having a standard naming to interoperate with modules is easier to share reusable “super classes” that manage application stacks / roles / components that use different modules for different applications from different authors.