About

About Icinga 2

What is Icinga?

Licensing

Support

What's New in Version 0.0.3

2. Getting Started

2.1 Installation

Configuration Syntax

Object Definition

Icinga 2 features an object-based configuration format. In order to define objects the *object* keyword is used:

```
object Host "host1.example.org" {
  display_name = "host1",

  macros = {
    address = "192.168.0.1"
  }
}
```

> **Note**
>
> The Icinga 2 configuration format is agnostic to whitespaces and new-lines.

> **Note**
>
> Colons (:) are not permitted in object names.

Each object is uniquely identified by its type (*Host*) and name (*host1.example.org*). Objects can contain a comma-separated list of property declarations. The following data types are available for property values:

Numeric Literals

A floating-point number.

Example:

```
-27.3
```

Duration Literal

Similar to floating-point numbers except for the fact that they support suffixes to help with specifying time durations.

Example:

```
2.5m
```

Supported suffixes include ms (milliseconds), s (seconds), m (minutes) and h (hours).

String Literals

A string.

Example:

```
"Hello World!"
```

Certain characters need to be escaped. The following escape sequences are supported:

Character |Escape sequence ——————————|—————————— " |\" \ |\\ <TAB> |\t <CARRIAGE-RETURN> |\r <LINE-FEED> |\n <BEL> |\b <FORM-FEED> |\f

In addition to these pre-defined escape sequences you can specify arbitrary ASCII characters using the backslash character (\) followed by an ASCII character in octal encoding.

Multiline String Literals

Strings spanning multiple lines can be specified by enclosing them in {{{ and }}}.

Example.

```
{{{This
is
a multi-line
string.}}}
```

Boolean Literals

The keywords *true* and *false* are equivalent to 1 and 0 respectively.

Null Value

The *null* keyword can be used to specify an empty value.

Dictionary

An unordered list of key-value pairs. Keys must be unique and are compared in a case-insensitive manner.

Individual key-value pairs must be separated from each other with a comma. The comma after the last key-value pair is optional.

Example:

```
{
  address = "192.168.0.1",
  port = 443
}
```

> **Note**
>
> Identifiers may not contain certain characters (e.g. space) or start with certain characters (e.g. digits). If you want to use a dictionary key that is not a valid identifier you can put the key in double quotes.

> **Note**
>
> Setting a dictionary key to null causes the key and its value to be removed from the dictionary.

Array

An ordered list of values.

Individual array elements must be separated from each other with a comma. The comma after the last element is optional.

Example:

```
[
  "hello",
  "world",
  42,
  [ "a", "nested", "array" ]
]
```

Note

An array may simultaneously contain values of different types, e.g. strings and numbers.

Operators

In addition to the = operator shown above a number of other operators to manipulate configuration objects are supported. Here's a list of all available operators:

Operator =

Sets a dictionary element to the specified value.

Example:

```
{
  a = 5,
  a = 7
}
```

In this example a has the value 7 after both instructions are executed.

Operator +=

Modifies a dictionary or array by adding new elements to it.

Example:

```
{
  a = [ "hello" ],
  a += [ "world" ]
}
```

In this example a contains both *"hello"* and *"world"*. This currently only works for dictionaries and arrays.

Attribute Shortcuts

Indexer Shortcut

Example:

```
{
  hello["key"] = "world"
}
```

This is equivalent to writing:

```
{
  hello += {
    key = "world"
  }
}
```

Inheritance

Objects can inherit attributes from other objects.

Example:

```
template Host "default-host" {
  check_interval = 30,

  macros["color"] = "red"
}

template Host "test-host" inherits "default-host" {
  macros["color"] = "blue"
}

object Host "localhost" inherits "test-host" {
  macros["address"] = "127.0.0.1",
  macros["address6"] = "::1"
}
```

The *"default-host"* and *"test-host"* objects are marked as templates using the *template* keyword. Unlike ordinary objects templates are not instantiated at runtime. Parent objects do not necessarily have to be templates though in general they are.

Note

The final macros dictionary contains all 3 macros and the macro *color* has the value *"blue"*.

Parent objects are resolved in the order they're specified using the *inherits* keyword.

Variables

Global variables can be set using the *set* keyword:

```
set VarName = "some value"
```

The value can be a string, number, array or a dictionary.

Constant Expressions

Simple calculations can be performed using the constant expression syntax:

```
{
  check_interval = (15 * 60)
}
```

Valid operators include +, -, * and /. The default precedence rules can be overriden by grouping expressions using parentheses:

```
{
  check_interval ((15 * 60) / 2)
}
```

Global variables may be used in constant expressions.

```
set MyCheckInterval = 10m

...

{
  check_interval = (MyCheckInterval / 2.5)
}
```

Note

Constant expressions are evaluated as soon as they're encountered in the configuration file.

Comments

The Icinga 2 configuration format supports C/C++-style comments.

Example:

```
/*
 This is a comment.
 */
object Host "localhost" {
  check_interval = 30, // this is also a comment.
  retry_interval = 15
}
```

Includes

Other configuration files can be included using the *include* directive. Paths must be relative to the configuration file that contains the *include* directive.

Example:

```
include "some/other/file.conf"
include "conf.d/*.conf"
```

> **Note**
>
> Wildcard includes are not recursive.

Icinga also supports include search paths similar to how they work in a C/C++ compiler:

```
include <itl/itl.conf>
```

Note the use of angle brackets instead of double quotes. This causes the config compiler to search the include search paths for the specified file. By default $PREFIX/icinga2 is included in the list of search paths.

Wildcards are not permitted when using angle brackets.

Library directive

The *library* directive can be used to manually load additional libraries. Libraries can be used to provide additional object types and methods.

Example:

```
library "snmphelper"
```

Note

The *icinga* library is automatically loaded at startup.

Global Variables ==========

IcingaPrefixDir

Read-only. Contains the installation prefix that was specified with ./configure –prefix. Defaults to /usr/local

IcingaLocalStateDir

Read-only. Contains the path of the local state directory. Defaults to IcingaPrefixDir + "/var".

IcingaPkgLibDir

Read-only. Contains the path of the package lib directory. Defaults to IcingaPrefixDir + "/lib/icinga2".

IcingaPkgDataDir

Read-only. Contains the path of the package data directory. Defaults to IcingaPrefixDir + "/share/icinga2".

IcingaStatePath

Read-write. Contains the path of the Icinga 2 state file. Defaults to IcingaLocalStateDir + "/lib/icinga2/icinga2.state".

IcingaPidPath

Read-write. Contains the path of the Icinga 2 PID file. Defaults to IcingaLocalStateDir + "/run/icinga2/icinga2.pid".

IcingaMacros

Read-write. Contains global macros. Not set by default.

Example:

```
set IcingaMacros = {
  plugindir = "/opt/check-plugins"
}
```

ApplicationType

Read-write. Contains the name of the Application type. Defaults to "IcingaApplication".

Object Types

ConsoleLogger

Specifies Icinga 2 logging to the console.

Example:

```
object ConsoleLogger "my-debug-console" {
  severity = "debug"
}
```

Attributes:

———————-|———————- severity |**Optional.** The minimum severity for this log. Can be "debug", "information", "warning" or "critical". Defaults to "information".

FileLogger

Specifies Icinga 2 logging to a file.

Example:

```
object FileLogger "my-debug-file" {
  severity = "debug",
  path = "/var/log/icinga2/icinga2-debug.log"
}
```

Attributes:

————————-|————————- path |**Required.** The log path. severity |**Optional.** The minimum severity for this log. Can be “debug”, “information”, “warning” or “critical”. Defaults to “information”.

SyslogLogger

Specifies Icinga 2 logging to syslog.

Example:

```
object SyslogLogger "my-crit-syslog" {
  severity = "critical"
}
```

Attributes:

————————-|————————- severity |**Optional.** The minimum severity for this log. Can be “debug”, “information”, “warning” or “critical”. Defaults to “information”.

CheckCommand

A check command definition. Additional default command macros can be defined here.

Example:

```
object CheckCommand "check_snmp" inherits "plugin-check-command" {
  command = "$plugindir$/check_snmp -H $address$ -C $community$ -o $oid$",

  macros = {
    address = "127.0.0.1",
    community = "public",
  }
}
```

NotificationCommand

A notification command definition.

Example:

```
object NotificationCommand "mail-service-notification" inherits "plugin-notification-commanc
  command = [
    "/opt/bin/send-mail-notification",
    "$CONTACTEMAIL$",
    "$NOTIFICATIONTYPE$ - $HOSTNAME$ - $SERVICEDESC$ - $SERVICESTATE$",
    {{{***** Icinga  *****

Notification Type: $NOTIFICATIONTYPE$

Service: $SERVICEDESC$
Host: $HOSTALIAS$
Address: $HOSTADDRESS$
State: $SERVICESTATE$

Date/Time: $LONGDATETIME$

Additional Info: $SERVICEOUTPUT$

Comment: [$NOTIFICATIONAUTHORNAME$] $NOTIFICATIONCOMMENT$}}}
  ]
}
```

EventCommand

An event command definition.

> **Note**
>
> Similar to Icinga 1.x event handlers.

Example:

```
object EventCommand "restart-httpd-event" inherits "plugin-event-command" {
  command = "/opt/bin/restart-httpd.sh",
}
```

Service

Service objects describe network services and how they should be checked by Icinga 2.

> **Best Practice**
>
> Rather than creating a *Service* object for a specific host it is usually easier to just create a *Service* template and using the *services* attribute in the *Host* object to associate these templates with a host.

Example:

```
object Service "localhost-uptime" {
  host = "localhost",
  short_name = "uptime",

  display_name = "localhost Uptime",

  check_command = "check_snmp",

  macros = {
    community = "public",
    oid = "DISMAN-EVENT-MIB::sysUpTimeInstance"
  }

  check_interval = 60s,
  retry_interval = 15s,

  servicegroups = [ "all-services", "snmp" ],
}
```

Attributes:

———————-|———————- host |**Required.** The host this service belongs to. There must be a *Host* object with that name. short_name |**Required.** The service name. Must be unique on a per-host basis (Similar to the service_description attribute in Icinga 1.x). display_name |**Optional.** A short description of the service. macros |TODO check_command |**Required.** The name of the check command. max_check_attempts|TODO check_period |TODO check_interval |**Optional.** The check interval (in seconds). retry_interval |**Optional.** The retry interval (in seconds). This is used when the service is in a soft state. Defaults to 1/5th of the check interval if not specified. event_command |TODO flapping_threshold|TODO volatile |TODO host_dependencies|TODO service_dependencies|TODO groups |**Optional.** The service groups this service belongs to. notifications |TODO

ServiceGroup

A group of services.

Example:

```
object ServiceGroup "snmp" {
  display_name = "SNMP services",
}
```

Attributes:

————————-|————————- display_name |**Optional.** A short description of the service group.

Notification

TODO

Example:

```
TODO
```

Attributes:

————————-|————————- host |TODO service |TODO macros |TODO users |TODO user_groups |TODO times |TODO notification_command|TODO notification_interval|TODO notification_period|TODO notification_type_filter|TODO notification_state_filter|TODO

User

TODO

Example:

```
TODO
```

Attributes:

————————-|————————- display_name |TODO macros |TODO groups |TODO enable_notifications|TODO notification_period|TODO notification_type_filter|TODO notification_state_filter|TODO

UserGroup

TODO

Example:

```
TODO
```

Attributes:

————————-|————————- display_name |TODO

TimePeriod

TODO

Example:

```
TODO
```

Attributes:

————————-|————————- display_name |TODO methods |TODO ranges |TODO

TimePeriod

TODO

Example:

```
TODO
```

Attributes:

————————-|————————- display_name |TODO

Domain

TODO

Example:

```
TODO
```

Attributes:

————————-|————————- acl |TODO

Host

A host.

> **Note**
>
> Unlike in Icinga 1.x hosts are not checkable objects in Icinga 2.

Example:

```
object Host "localhost" {
  display_name = "The best host there is",

  groups = [ "all-hosts" ],

  check = "ping",

  host_dependencies = [ "router" ],

  service_dependencies = [
    { host = "db-server", service = "mysql" }
  ],

  services["ping"] = {
    templates = [ "ping" ]
  },

  services["http"] = {
    templates = [ "my-http" ],

    macros = {
      vhost = "test1.example.org",
      port = 81
    }
  }
}
```

Attributes:

————————-|————————- display_name |**Optional.** A short description of the host. check |**Optional.** A service that is used to determine whether the host is up or down. This must be a service short name of a service that belongs to the host. groups |**Optional.** A list of host groups this host belongs to. host_dependencies|**Optional.** An array of host names which this host depends on. These dependencies are used to determine whether the host is unreachable. service_dependencies|**Optional.** An array of service names which this host depends on. Each array element must be a dictionary containing the keys "host" and "service". These dependencies are used to determine whether the host is unreachable. services |**Optional.** Inline definition of services. Each dictionary item specifies a service.The *templates* attribute can be used to specify an array of templates that should be inherited by the service.The new service's name is "hostname:service" - where "service" is the dictionary key in the services dictionary.The dictionary key is used as the service's short name. macros |TODO

HostGroup

A group of hosts.

Example:

```
object HostGroup "my-hosts" {
  display_name = "My hosts",
}
```

Attributes:

———————-|———————- display_name |**Optional.** A short description of the host group.

PerfdataWriter

Writes check result performance data to a defined path using macro pattern.

Example

```
object PerfdataWriter "pnp" {
  perfdata_path = "/var/spool/icinga2/perfdata/service-perfdata",

  format_template = "DATATYPE::SERVICEPERFDATA\tTIMET::$TIMET$\tHOSTNAME::$HOSTNAME$\tSERVIC

  rotation_interval = 15s,
}
```

Attributes:

———————-|———————- perfdata_path |**Optional.** Path to the service perfdata file. Defaults to IcingaLocalStateDir + "/cache/icinga2/perfdata/perfdata". format_template|**Optional.** Format template for the perfdata file. Defaults to a template that's suitable for use with PNP4Nagios. rotation_interval|**Optional.** Rotation interval for the file specified in *perfdata_path*. Defaults to 30 seconds.

> **Note**
>
> When rotating the perfdata file the current UNIX timestamp is appended to the path specified in *perfdata_path* to generate a unique filename.

IdoMySqlConnection

IDO DB schema compatible output into MySQL database.

Example:

```
library "db_ido_mysql"

object IdoMysqlConnection "mysql-ido" {
  host = "127.0.0.1",
  port = 3306,
  user = "icinga",
  password = "icinga",
  database = "icinga",
  table_prefix = "icinga_",
  instance_name = "icinga2",
  instance_description = "icinga2 dev instance"
}
```

Attributes:

————————-|————————- host |**Optional.** MySQL database host address. Defaults to “localhost”. port |**Optional.** MySQL database port. Defaults to 3306. user |**Optional.** MySQL database user with read/write permission to the icinga database. Defaults to “icinga”. password |**Optional.** MySQL database user’s password. Defaults to “icinga”. database |**Optional.** MySQL database name. Defaults to “icinga”. table_prefix |**Optional.** MySQL database table prefix. Defaults to “icinga_”. instance_name |**Optional.** Unique identifier for the local Icinga 2 instance. Defaults to “default”. instance_description|**Optional.** Description for the Icinga 2 instance.

LiveStatusListener

Livestatus API interface available as TCP or UNIX socket.

Example:

```
library "livestatus"

object LivestatusListener "livestatus-tcp" {
  socket_type = "tcp",
  bind_host = "127.0.0.1",
  bind_port = "6558"
}

object LivestatusListener "livestatus-unix" {
```

```
  socket_type = "unix",
  socket_path = "/var/run/icinga2/livestatus"
}
```

Attributes:

———————-|———————- socket_type |**Optional.** Specifies the socket type. Can be either “tcp” or “unix”. Defaults to “unix”. bind_host |**Optional.** Only valid when socket_type is “tcp”. Host address to listen on for connections. Defaults to “127.0.0.1”. bind_port |**Optional.** Only valid when *socket_type* is “tcp”. Port to listen on for connections. Defaults to 6558. socket_path |**Optional.** Only valid when *socket_type* is “unix”. Specifies the path to the UNIX socket file. Defaults to IcingaLocalStateDir + “/run/icinga2/livestatus”.

> **Note**
>
> UNIX sockets are not supported on Windows.

StatusDataWriter

TODO

Example:

```
TODO
```

Attributes:

———————-|———————- status_path |TODO objects_path |TODO

ExternalCommandListener

TODO

Example:

```
TODO
```

Attributes:

———————-|———————- command_path |TODO

CompatLogger

TODO

Example:

```
TODO
```

Attributes:

———————-|———————- log_dir |TODO rotation_method|TODO

CheckResultReader

TODO

Example:

```
TODO
```

Attributes:

———————-|———————- spool_dir |TODO

CheckerComponent

TODO

Example:

```
library "checker"

object CheckerComponent "checker" { }
```

NotificationComponent

TODO

Example:

```
library "notification"

object NotificationComponent "notification" { }
```

ClusterListener

TODO

Example:

```
TODO
```

Attributes:

————————-|————————- cert_path |TODO ca_path |TODO bind_host |TODO bind_port |TODO peers |TODO

Endpoint

Endpoint objects are used to specify connection information for remote Icinga 2 instances.

Example:

```
library "cluster"

object Endpoint "icinga-c2" {
  node = "192.168.5.46",
  service = 7777,
}
```

Attributes:

————————-|————————- node |**Required.** The hostname/IP address of the remote Icinga 2 instance. service |**Required.** The service name/port of the remote Icinga 2 instance. config_files |TODO accept_config |TODO