

Cfengine Nova

Enterprise cfengine
for version 1.0.0

1 Introduction to Nova

Cfengine Enterprise editions are commercially licensed versions of the core Cfengine major version 3, with build-time library extensions.

Cfengine enterprise features are designed to both extend and simplify the use of Cfengine in enterprise scenarios, ultimately providing a single framework for self-healing, hands-free automation with integrated knowledge management. Each extension has been carefully designed to meet a specific need, replacing cumbersome and insecure technologies currently available in datacentre products.

Each edition may be described as a Commercial Open Source product. Users have full access to the source code and are free to modify it without warranty. Changes may be used within the organization or shared with other license-holders. Changes may also be contributed back for use by other license holders in future releases.

You should use this brief guide to CfNova's features in concert with the Cfengine Reference Manual, available from the Cfengine website, or in the 'docs' directory.

1.1 Installing Cfengine Nova

You can choose to build Cfengine Nova directly from source, or install packaged binaries provided by Cfengine.

1.1.1 Building cfengine yourself

Building from source allows you to tailor the build of cfengine and its dependencies to your specific environment. In order to install cfengine, you should first ensure that the following packages are installed.

OpenSSL Open source Secure Sockets Layer for encryption.
URL: <http://www.openssl.org>

BerkeleyDB (version 3.2 or later)
Light-weight flat-file database system.
URL: <http://www.oracle.com/technology/products/berkeley-db/index.html>

In addition...

A fully featured Cfengine Nova requires

Graphviz The Graphviz library will be required in the short term for plotting dependency diagrams. The library will be deprecated in the future.

Perl Compatible Regular Expressions

The PCRE library is used for powerful pattern matching features. It is considerably superior to the default POSIX library.

Open LDAP

The Open LDAP library is needed for LDAP integration functions.

MySQL and/or PostgreSQL client libraries

Database communication requires access to the APIs for these databases.

On Windows machines, you need to install the basic Cygwin DLL from <http://www.cygwin.com> in order to run cfengine.



Additional functionality becomes available if other libraries are present, e.g. OpenLDAP, client libraries for MySQL and PostgreSQL, etc. It is possible to run cfengine without these, but related functionality will be missing.

Unless you have purchased ready-to-run binaries, or are using a package distribution, you will need to compile cfengine. For this you will also need a build environment tools: `gcc`, `flex`, `bison`.

The preferred method of installation is then

```
tar xzf cfengine-x.x.x.tar.gz
cd cfengine-x.x.x
./configure
make
make install
```

This results in binaries are installed in `/usr/local/sbin` and helper files are installed in `/usr/share/doc`.

1.2 Installing from packages

TODO

1.3 Enhancements

The enhancement provide by Cfengine Nova fall into three main categories:

Productivity enhancements:

Feature that are designed to make it easier to work with cfengine on a day to day basis. For example, the ability to look up syntax on the fly from the web or the command line. Using the tools for database interaction, the process of setting up a cfengine knowledge base at your organization has been greatly simplified.

LDAP query functions have been added for integration with LAP or Active Directory information.

Continuity and repair related enhancements:

These include the ability to interact with popular SQL databases (currently MySQL and PostgreSQL), as well as embedded system databases such as the Microsoft System Registry, defining, validating, scanning or repairing their tabular structure and the data within them.

Access Control List (ACL) support for Linux is now added for pinpoint accuracy in file permission security. This will soon join the support for Posix ACLs on solaris and NTFS ACLs on Windows already present in cfengine 2. Cfengine Nova's ACL support includes a completely new generic cfengine model for ACLs that will translate across multiple platforms so that users can as closely as possible translate identical requirements across multiple platforms with different implementations. Native ACLs are also supported.

Reporting enhancements:

As the first step toward a full CMDB solution that will be part of Cfengine Constellation, Nova can provide a wider range of system reports about information collected by cfengine on performance, security and state. This includes the ability to perform custom system discovery, and log the data into a variety of special reports. All discovery and measurement of the system is provided through cfengine's lightweight custom monitoring capabilities.



Data from cfengine can be extracted in HTML, XML and text formats (e.g. CSV) for easy integration with other presentation tools, or direct viewing. The reports can be collected and post-processed with Cfengine Constellation for an enterprise level overview of global networks.

As part of policy-writing, Cfengine Nova allows you to track dependencies of policy items through its self-documenting features. The knowledge agent can then take this information, analyze it and present it as part of an overview of the system.

2 Productivity and Documentation

2.1 Syntax lookup on the command line

An annoying aspect of any software is the need to browse through a manual to find quick answers to questions. Ideally one would only look at manuals during a learning phase; thereafter we want to see examples and summaries of what we basically already know but cannot keep in our heads.

Users differ in their habits. Most Unix system administrators prefer to work in the command line and find the need to go to a manual a distraction. Indeed, if the network is non-functional or we are working off-line, instant command line help is a great bonus.

Other users prefer the point and click of a familiar web interface. Cfengine Nova provides both these options to users to provide quick answers.

Of the two, a web interface is clearly the most flexible. Far more information can be browsed on the web than is practical with a simple text interface. However, of the two, the command line interface is by far the fastest way to get answers to simple questions of syntax.

Cfengine's knowledge agent knows cfengine's syntax tree and can summarize it at the keyword level.

```
atlas$ ./cf-know --syntax link_from
Constraint link_from (of promise type files) has possible values:

link_from ~ defined in a separate body, with elements

source      ~ ()
link_type    ~ (symlink,hardlink,relative,absolute,none)
copy_patterns ~ ()
when_no_source ~ (force,delete,nop)
link_children ~ (true,false,yes,no,on,off)
when_linking_children ~ (override_file,if_no_such_file)
```

Description: A set of patterns that should be copied and synchronized instead of linked

```
atlas$ ./cf-know -S acl_type
Body constraint acl_type is part of acl (in promise type files) and has possible values:
```

```
acl_type ~ (posix,ntfs)
```

Description: Access control list type for the affected file system

2.2 Knowledge map creation

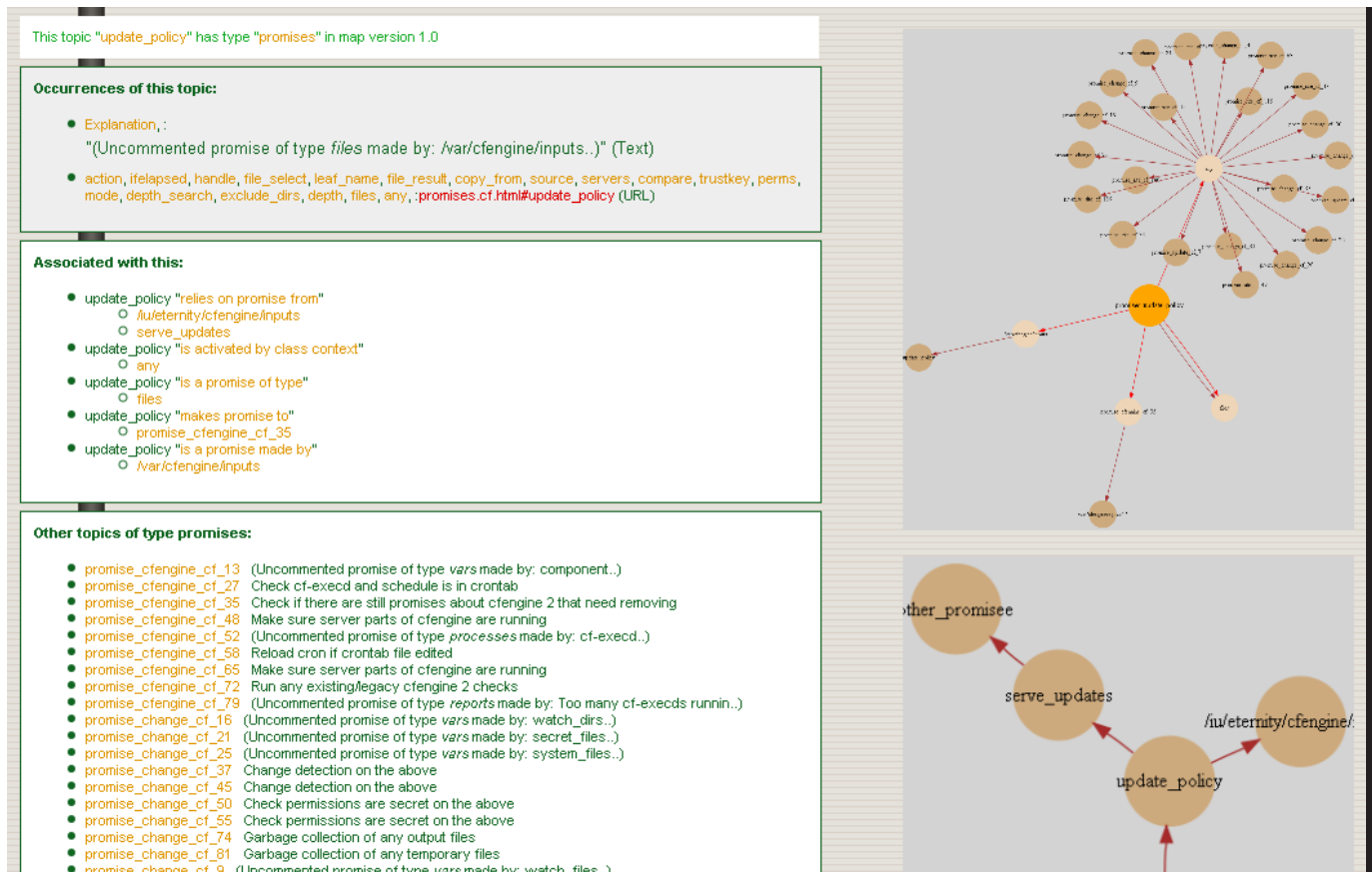
Cfengine helps you to make sense of the knowledge in your policy, by making a knowledge map – and then extend it with your own enterprise information.

In the example figure below we see typical page generated by the knowledge agent about a cfengine promise. In this case the promise has been given the handle `update_policy`, and the associations and the lower graph shows how this promise relates to other promises through its documented dependencies (these are documented from the `promisees` and `depends_on` attributes of other promises.).

The upper figure shows the thirty nearest topics (of any kind) that are related to this one. Here the relationships are unspecific. This diagram can reveal pathways to related information that are often

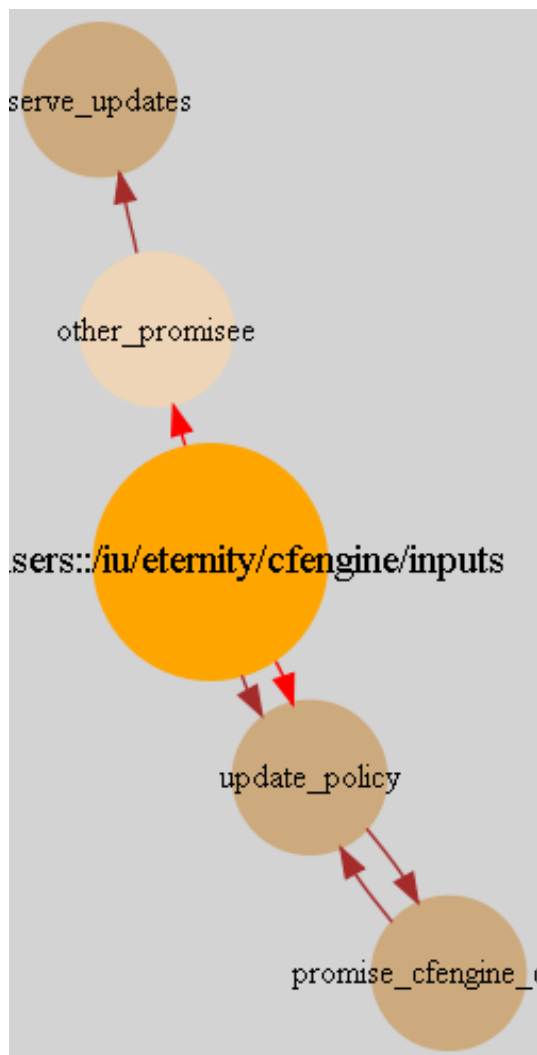


unexpected, and illustrates relationships that broaden one's understanding of the place the current promise occupies within the whole.




The second figure shows a blow up of another dependency illustration. Although these illustrations are just renderings of the associations shown in text, they are useful for visualizing several levels of depth in the associative network. In particular, one can see the other promises that could be affected if we were to make a change to the current promise.

Such impact analyses can be crucial to planning change and release management of policy. ITIL best practices recommend thorough pre-analyses of release changes. Cfengine Nova combined with good practice of documentation makes this straightforward.



Another example topic page.


CFENGINE knowledge console

promises::serve_backup

This topic "serve_backup" has type "promises" in map version 1.0

Occurrences of this topic:

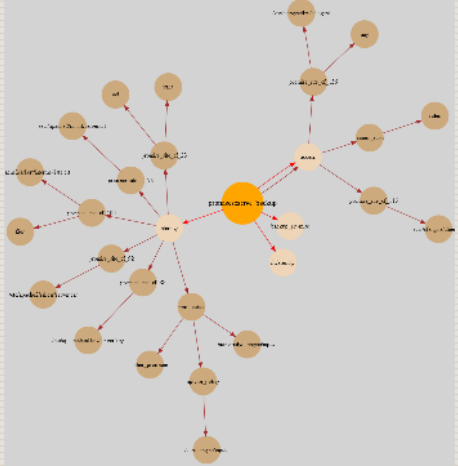
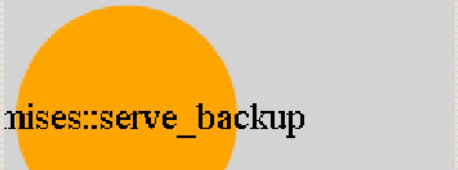
- Explanation:
 - "(Uncommented promise of type *access* made by: */iu/eternity..*)" (Text)
- handle, admit, maproot, eternity, access, :promises.cf.html#serve_backup (URL)

Associated with this:

- serve_backup "is activated by class context"
 - eternity
- serve_backup "is a promise of type"
 - access
- serve_backup "is a promise made by"
 - /iu/eternity
- serve_backup "makes promise to"
 - backup_promise

Other topics of type promises:

- promise_cfengine_cf_13 (Uncommented promise of type *vars* made by: component..)
- promise_cfengine_cf_27 Check cf-execd and schedule is in crontab
- promise_cfengine_cf_35 Check if there are still promises about cfengine 2 that need removing
- promise_cfengine_cf_48 Make sure server parts of cfengine are running
- promise_cfengine_cf_52 (Uncommented promise of type *processes* made by: cf-execd..)
- promise_cfengine_cf_58 Reload cron if crontab file edited
- promise_cfengine_cf_65 Make sure server parts of cfengine are running
- promise_cfengine_cf_72 Run any existing/legacy cfengine 2 checks
- promise_cfengine_cf_79 (Uncommented promise of type *reports* made by: Too many cf-execs runnin..)
- promise_change_cf_16 (Uncommented promise of type *vars* made by: watch_dirs..)
- promise_change_cf_21 (Uncommented promise of type *vars* made by: secret_files..)
- promise_change_cf_25 (Uncommented promise of type *vars* made by: system_files..)
- promise_change_cf_37 Change detection on the above

A knowledge base is an implementation of a Topic Map which is an ISO standard technology. A topic map works like an index that can point to many different kinds of external resources, and may contain simple text and images internally. So you use it to bind together documents of any kind.

A cfengine knowledge base is not a new document format, it is an overlay map that joins ideas and resources together, and displays relationships.

2.2.1 Procedure for generating a local knowledge map

You will need a computer running a web server, with some active server page technology to use as a wrapper for cfengine. Our example assumes that the wrapper will be a PHP enabled web server and a backend database (for cfengine not PHP). The wrapper runs `cf-know`, which renders pages for a web-browser. In order to work, your cfengine build needs to have been compiled with database support. Currently MySQL and PostgreSQL are supported.

If you are building from source, the default location for basic set-up files is `/usr/local/share/doc/cfnova/knowledge/`. If your Nova software is pre-installed by an operating system vendor, this will lie in `/usr/share/doc/cfnova/knowledge/`. In this directory you will find all of the files to get started.

In this directory, you will find a file called `knowledge.cf` which automates the update of the knowledge base (see below).

To bootstrap the knowledge base:

1. Copy the files provided in the 'knowledge' directory to a web-enabled directory (document root).
2. you might want to consider using version control for the knowledge map data as you will be maintaining any edits you make by hand, perhaps as a team.
3. Edit the 'index.php' file to reflect the path for the directory you just decided on and the location of cf-know.

```
# Set this to your local knowledge base
```

```
$reports = "/usr/local/share/doc/cfnova/knowledge/";
$file = "/usr/local/share/doc/cfnova/knowledge/enterprise_run.cf";
```

4. Edit the file 'enterprise_control.cf'. You will need to set up the parameters for a database that cfengine will use to cache the topic map. Cfengine Nova supports MySQL and PostgreSQL databases currently. You do not need modules for these databases in your PHP installation, as cfengine talks to the database directly, but cfengine must have been compiled with database support.

```
body knowledge control
```

```
{
# Decide the name of a local database

sql_database => "cf_knowledge_map";
sql_owner => "root";
sql_passwd => "";
sql_type => "mysql";
sql_connection_db => "mysql_manage_point";
#sql_server => "otherhost";
```

If you create the database for cf-know by hand, then remember to grant access to the owner as specified above or cf-know will not be able to add data.

If you want cf-know to create the database for you, make sure there is some database 'mysql_manage_point' that cf-know has permissions to connect to. We recommend letting cf-know create the database, then you only need to decide on its name sql_database;

5. Finally, build a view of your configuration. You must update this part each time that you change your master-configuration.

```
host$ cd /usr/local/share/doc/cfnova/knowledge
host$ cf-promises -r
```

This generates the files 'promise_knowledge.cf', 'promises.cf.html' and 'promise_output_common.html' that are used by cf-know in creating the policy overview.

Make sure all these files are readable by the web-server.

The file 'promise_knowledge.cf' contains a knowledge bundle. If Nova has not been installed correctly, you will see an empty bundle.



```
bundle knowledge CfengineSiteConfiguration
{
}
```

Cfengine Nova fills this bundle with extensively cross referenced knowledge about Cfengine syntax and about your currently active policy.

6. When all of the files are assembled, we use the knowledge agent to populate the database.

7.

```
cf-know -f ./enterprise.cf -s
```

2.2.2 The 'knowledge.cf' file

The bundled configuration file 'knowledge.cf' can be included in your cfengine bundlesequence to automate the updating of the knowledge map. You should copy this file to your master 'inputs' directory and add the bundle 'setup_knowledge' to the bundle sequence. This will ensure that cf-promise and cf-know are run when policy updates are made, to update the database.

In this file you should enter the name of the document root on the web-server that will run the knowledge base. This means that you need to edit three files in all to set up the knowledge base, each representing different aspects of this process.

'index.php'

This is for the web server.

'enterprise_control.cf'

This is for the cf-know rendering of the knowledge executed by the web-server.

'knowledge.cf'

This is for building and updating the database that is read by web-renderer.

2.2.3 Trouble shooting the knowledge base

Cfengine Nova will try to create the database and all of the tables automatically. If this does not happen, the likely explanation is that it does not have permission from the database server to do this.

Connect to the database and grant access to it, e.g.

```
mysql
USE mysql_manage_point
GRANT ALL on * to root;
or
psql postgres_manage_point
GRANT ALL on * to root;
@end
```

@node Understanding dependencies, , Knowledge map creation, Productivity and Documentation■
 @section Understanding dependencies

Nova adds automatically instrumentation to the policy compilation that allows you to trace dependencies, provided you follow the best-practice when constructing your promise rules.



When writing promises, get into the habit of giving every promise a comment that explains its intention.

Also, give related promises `@i{handles}`, or labels that can be used to refer to them by.

`@verbatim`

`files:`

```
"/var/cfengine/inputs"

handle => "update_policy",

perms => system("600"),
copy_from => mycopy("${master_location}", "${policy_server}"),
depth_search => recurse("inf"),
file_select => input_files,
action => immediate;
```

If a promise affects another promise in some way, you can make the affected promise one of the promisees, like this:

`access:`

```
"/master/cfengine/inputs" -> { "update_policy", "other_promisee" },

handle => "serve_updates",

admit => { "217.77.34.*" };
```

Conversely, if a promise might depend on another in some (even indirect) way, document this too.

`files:`

```
"/var/cfengine/inputs"

handle      => "update_policy",
depends_on => "serve_updates",

perms => system("600"),
copy_from => my-copy("${master_location}", "${policy_server}"),
depth_search => recurse("inf"),
file_select => input_files,
```



```
action => immediate;
```

The lower of the two diagrams on the knowledge page shows graphically how dependency trails flow. The text links show the specific of each relationship.

- When a bundle is passed a parameter by another
- In method calls, a passed variable value is considered a causal influence.
- When a variable is used in a shell command
- When a variable is set by a function call that retrieves from a database/file

3 Monitoring extensions

Cfengine's monitoring component `cf-monitord` records a number of performance data about the system by default. These include process counts, service traffic, load average and CPU utilization and temperature when available.

Cfengine Nova extends this in two ways. First it adds a three year trend summary based any 'shift'-averages. Second, it add customizable promises to monitor or log specific quite specific user data through a generic interface. The end result is to either generate a periodic time series, like the above mentioned values, or to log the results to custom-defined reports.

3.1 Long term trends

Cfengine normally operates with time-series data represented in two forms:

- A weekly average, plotted on a periodogram, showing performance now in relation to the same time of week in previous weeks. After about a month data are forgotten to ensure a sufficient rate of adaptation to new patterns.
- The past four hours in high resolution.

Cfengine Nova adds quarter day averages of recorded time-series which go back three years in time. Three years is considered to be the lifetime of a computer. Summaries of the detailed performance are summarized by flat averages for a four-shift day: from midnight 00:00 to 06:00 (night shift), from 06:00 to 12:00 (morning shift), from 12:00 to 18:00 (after-noon), and from 18:00 to 00:00 (evening-shift).

The extensions here form the basis of data collection and discovery probes for the CMDB solution in Cfengine Constellation. With these probes you effectively have a sophisticated, distributed CMDB. Data may be extracted from the embedded learning database using `cf-report`. In Nova, the multi-year reports are generated automatically along with the others when time-series data are requested.

3.2 Custom promises to measure

Cfengine Nova adds a new promise type in bundles for the monitoring agent. These are written just like all other promises within a bundle destined for the agent concerned. In this case:

```
bundle monitor watch
```

```
{
measurements:

    # promises ...
}
```

3.2.1 Extraction strings and logging

Let's take a generic example. Suppose we have a file of nonsense `/tmp/testmeasure` and we want to extract some information that we call a 'blonk' from the file. A blonk is always on the second line of this file following a prefix 'Blonk blonk '. We would get the value like this:

```
"/tmp/testmeasure"
```



```

    handle => "blonk_watch",
    stream_type => "file",
    data_type => "string",
    history_type => "log",
    units => "blonks",
    match_value => find_blonks,
    action => sample_min("10");

```

This promise body has several attributes.

handle It is essential to give measurement promises handles, as these are used to label the log files that will store the values.

stream_type Tells us that we are reading from what the system considered to be a regular file on the filesystem.

data_type This says that data are to be treated as text with no other meaning.

history_type This tells us that we want to log the values with a time stamp.

units This string is used in documentation to explain the measurement units of this result.

match_value This is a body reference that represents the algorithm by which we extract data from the file.

action This is the generic action parameter that may be added to all promises. We use it here to limit the sample rate of this promise; `cf-monitord` samples by default at a rate of once per 2.5 minutes.

The matching body uses a method for selecting the correct line, and a way for extracting a pattern from the line. In every case the value extracted is described by using a regular expression *back-reference*, i.e. a parenthesized expression within a regular expression. The expression should match the entire line and should contain exactly one parenthesis.

```

body match_value find_blonks
{
    select_line_number => "2";
    extraction_regex => "Blonk blonk ([blonk]+).*";
}

```

The sampling rate is controlled by using the generic `action` constraint.

```

body action sample_min(x)
{
    ifelapsed => "$(x)";
    expireafter => "$(x)";
}

```

3.2.2 Extracting one-off numerical data

In this example we extract an integer value from an existing file. Notice that `cfengine` samples the process table during processes promises so you might be able to save a new execution of a shell

command and use the cached data, depending on your need for immediacy. It is always good practice to limit the system load incurred by monitoring.

```
# Test 2 - follow a special process over time
# using cfengine's process cache to avoid resampling

"/var/cfengine/state/cf_rootprocs"

    handle => "monitor_self_watch",
    stream_type => "file",
    data_type => "int",
    history_type => "none",
    units => "kB",
    match_value => proc_value(".*cf-monitor.*",

        "root\s+[0-9.]+\s+[0-9.]+\s+[0-9.]+\s+[0-9.]+\s+([0-9]+).*");
```

This match body selects a line matching a particular regular expression and extracts the 6th column of the process table. The regular expression skips first the root string and then five numerical values. The value is extracted into a one-off value

```
body match_value proc_value(x,y)
{
    select_line_matching => "$(x)";
    extraction_regex => "$(y)";
}
```

3.2.3 Extraction to list variable

In this example we discover a list of disks attached to the system.

```
# Test 3, discover disk device information

"/bin/df"

    stream_type => "pipe",
    data_type => "slist",
    history_type => "static",
    units => "device",
    match_value => file_system,
    action => sample_min("480"); # this is not changing much!
```

```
body match_value file_system
{
    select_line_matching => "/*.*";
    extraction_regex => "(.*)";
}
```

3.3 Uses for custom monitoring

Unlike most other monitoring tools that use heavy-weight scripting languages to exact data, often running many processes for each measurement, cfengine is a lightweight probe, using file interfaces and regular expressions to extract data. Thus its impact on the system is minimal. The possibilities for using this are therefore extremely broad:

- Extracting accounting data from systems for charge-back. This could be useful in cloud scenarios.
- Discovering memory leaks.
- Looking for zombie processes relating to specific software.
- Logging uptime.
- System class-dependent discovery and extraction of any kind of text for insertion into a CMDB.

4 Databases

Cfengine Nova can interact with commonly used database servers to keep promises about the structure and content of data within them.

Cfengine is, of course, not the tool of choice for performing complex database transactions, but it is an appropriate choice of tool for managing policy about the existence and structure of databases and correcting discrepancies.

4.1 How to manage databases

There are two main cases of database management to address: small embedded databases and large centralized databases.

Cfengine is a tool whose strength lies distributed management of computers. Databases are often centralized entities that have single point of management, so a large monolithic database is more easily managed with other tools. However, cfengine can still monitor changes and discrepancies, and it can manage smaller embedded databases that are distributed in nature, whether they are SQL, registry or future types.

So creating 100 new databases for test purposes is a task for cfengine, but adding a new item to an important production database is not a task that we recommend using cfengine for.

There are three kinds of database supported by Nova:

LDAP - The Lightweight Directory Access Protocol

A hierarchical network database primarily for reading simple schema.

SQL - Structured Query Language

A number of relational databases (currently supported: MySQL, Postgres) for reading and writing complex data.

Registry - Microsoft Registry

An embedded database for interfacing with system values in Microsoft Windows.

In addition, cfengine uses a variety of embedded databases for its own internals.

4.2 Database access rights

Cfengine's ability to make promises about databases depends on the good grace of the database server. Embedded databases are directly part of the system and promises can be made directly. However, databases running through a server process (either on the same host or on a different host) are independent agents and cfengine cannot make promises on their behalf, unless they promise (grant) permission for cfengine to make the changes. Thus the pre-requisite for making SQL database promises is to grant a point of access on the server.

4.3 Creating a point of contact on a server

The default behaviour is for databases to deny connection access to external parties. However, it is possible to create a connection database that can be used as an anchor point. Once cfengine is connected to any database, it can create new ones.



4.4 Creating SQL databases

Although the Structured Query Language (SQL) is a standard, database servers diverge wildly in their approaches to interaction. Cfengine currently supports two open source database implementations: MySQL and PostgreSQL and these two could not be more different.

Although it would be desirable to perform operations like create and destroy databases from within cfengine, these operations can be difficult to set up due to security design of the database servers. This is where cfengine's idea of *voluntary cooperation* helps us to understand why. The database service (even running on localhost) is an independent entity that we must interact with through its service portal. It promises nothing a priori, so we must arrange for the necessary promises to be kept so that the database will respond to cfengine's manipulations. In other words, we have to grant access.

4.4.1 Creating a database manually

```
body common control
{
bundlesequence => { "databases" };
}

bundle agent databases
{
# Create database first - for postgres we could use a command

commands:

    "/usr/bin/createdb cf_topic_maps",

        contain => as_user("postgres");

# Create a table in the new database

databases:

    "cf_topic_maps/topics"

        database_operation => "create",
        database_type => "sql",
        database_columns => {
            "topic_name,varchar,256",
            "topic_comment,varchar,1024",
            "topic_id,varchar,256",
            "topic_type,varchar,256",
            "topic_extra,varchar,26"
        },

        database_server => myserver;
```

```

}

#####

body database_server myserver
{
  db_server_owner => "postgres";
  db_server_password => "";
  db_server_host => "localhost";
  db_server_type => "postgres";
}

#

body contain as_user(x)
{
  exec_owner => "$x";
}

```

4.4.2 Creating a database directly

Both MySQL and PostgreSQL provide default databases of the same name to connect to as an anchor point, allowing further databases to be created thereafter. In fact any database will do, you must just make sure that cfengine has access rights to connect to the database.

```

body common control
{
  bundlesequence => { "databases" };
}

bundle agent databases
{
  # Create table topics in database cf_topic_maps

  "cf_topic_maps"

  database_operation => "create",
  database_type => "sql",
  database_server => myserver("mysql");
}

```

```
#####
```

```
body database_server myserver(x)
{
db_server_owner => "$(x)";
db_server_password => "";
db_server_host => "localhost";
db_server_type => "$(mysql)";
db_server_connection_db => "$(x)";
}
```

```
body contain as_user(x)
{
exec_owner => "$(x)";
}
```

4.5 Database table promises

To refer to tables in an SQL database, you treat the tables as if they were subdirectories of the database. In this example, we create one of the topic map tables for the knowledge base.

Table existence is determined by the `database_operation` attribute. We specify columns in a table as doublets or triplets with SQL type-names. In this example cfengine will assume the promise below to be a complete specification of the table. Thus

- It will create non-existing columns (in no guaranteed order).
- It will report about columns that exist but are not promised, i.e. indicating that the table has been edited inappropriately.
- It will verify that the existing column names match their specification.

```
bundle agent databases
```

```
{
databases:

    "cf_topic_maps/topics"

    database_operation => "create",
    database_type => "sql",
    database_columns => {
        "topic_name,varchar,256",
        "topic_comment,varchar,1024",
        "topic_id,varchar,256",
        "topic_type,varchar,256",
        "topic_extra,varchar,26"
    },

    database_server => myserver;
```

```

}

#####

body database_server myserver
{
any::
  db_server_owner => "postgres";
  db_server_password => "";
  db_server_host => "localhost";
  db_server_type => "postgres";
  db_server_connection_db => "postgres";
none::
  db_server_owner => "root";
  db_server_password => "";
  db_server_host => "localhost";
  db_server_type => "mysql";
  db_server_connection_db => "mysql";
}

body contain as_user(x)
{
exec_owner => "$x";
}

```

4.6 MS Registry functions

Another important database for Windows systems is the MS registry. The registry looks like a filesystem, with files inside directories. In MS nomenclature, the registry consists of *(value,data)* pairs inside "keys" which are like directories. Keys may contain sub-keys in a hierarchical manner.

Managing data in the registry is a key part of Windows administration.

- Important system settings are located in and hence controlled by the registry.
- Security and compliance guidelines often specify values for the registry.
- If registry data become corrupted, software can cease to function. Compliance with government standards such as the U.S. FDCC regulations involves registry compliance checking.
- Automated repair (self-healing) of Windows machines can involve registry repair.

Cfengine Nova has functions for

- Creating and deleting keys, and *(value,data)* pairs.
- Scanning values and performing change detection.
- Restore or repair damaged registry data its cache in real-time.

4.6.1 Creating a registry key

```
body common control
```



```

{
bundlesequence => { "registry" };
}

bundle agent registry

{
databases:

    windows::

        "HKEY_LOCAL_MACHINE\SOFTWARE\Cfengine AS"

        database_operation => "create",
        database_type => "ms_registry";
}

```

In this example we create a sub-key called 'Cfengine AS' inside the software key. Directory-like notation is used, as in the Windows registry editor. To create a sub-key of this new key, we simply add another link:

```

bundle agent registry
{
databases:

    windows::

        "HKEY_LOCAL_MACHINE\SOFTWARE\Cfengine AS\Cfengine"

        database_operation => "create",
        database_type => "ms_registry";
}

```

4.6.2 Creating a value-data pair

```

body common control
{
bundlesequence => { "registry" };
}

bundle agent registry

{
databases:

```




```

windows::

"HKEY_LOCAL_MACHINE\SOFTWARE\Cfengine AS\Cfengine"

  database_operation => "create",
  database_rows => { "value1,REG_SZ,new value 1", "value2,REG_SZ,new val 2"} ,
  database_type => "ms_registry";

}

```

Value-data pairs are in-fact value-type-data triplets, though the most common registry type is REG_SZ. Users are referred to the Microsoft Developer Network documentation on the registry for details (search for 'msdn registry').

4.6.3 Scanning and restoring the registry

Cfengine has the ability to scan a portion of the registry and cache its values for change detection and even later restoration. Since this is a resource consuming operation it should be performed only rarely, i.e. one should limit the policy to scan to special occasions.

The key question to ask is: when do I expect the current state of the registry to be a proper and true reflection of the 'correct state' of the system. This is not a trivial question to answer and it should be given careful thought before any talk of restoring the state of the registry based on such a scan.

Some values in the registry change often and one would like to ignore them entirely. The `registry_exclude` list allows you to make a list of regular expressions to match keys and value-names which are neither scanned nor restored.

```

bundle agent registry

{
databases:

windows:: # add a time qualifier to this class

"HKEY_LOCAL_MACHINE\SOFTWARE\Cfengine AS\Cfengine"

  database_operation => "cache",    # cache,restore

  registry_exclude => {
    ".*Windows.*CurrentVersion.*",
    ".*Touchpad.*",
    ".*Capabilities.FileAssociations.*",
    ".*Rfc1766.*" ,
    ".*Synaptics.SynTP.*",
    ".*SupportedDevices.*8086",
    ".*Microsoft.*ErrorThresholds"
  },

```

```
database_type      => "ms_registry";
```

Once one believes and trusts in a scan of the registry as a matter of policy, the values can be compared to this golden standard and restored if in error with the following kind of promise:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\Cfengine AS"
```

```
database_operation => "restore",
database_type      => "ms_registry";
```

Deletion or alteration of a key or value will now be swiftly repaired without the need to reboot the system.

4.7 LDAP integration

The Lightweight Directory Access Protocol (LDAP) is the basis of enterprise data centralization in many organizations using a variety of products from multiple vendors, including the MS Active Directory. Cfengine Nova supports special inbuilt functions for extracting data from LDAP repositories. The functions follow the general pattern of Open LDAP search functions.

4.7.1 Function ldapvalue

```
(string) ldapvalue(uri,dn,filter,name,scope,security)
```

This function retrieves a single field from a single LDAP record identified by the search parameters. The first matching value is taken.

ARGUMENTS:

- | | |
|------------|---|
| 'uri' | String value of the ldap server. e.g. "ldap://ldap.cfengine.com.no" |
| 'dn' | Distinguished name, an ldap formatted name built from components, e.g. "dc=cfengine,dc=com". |
| 'filter' | String filter criterion, in ldap search, e.g. "(sn=User)". |
| 'name' | String value, the name of a single record to be retrieved, e.g. uid. |
| 'scope' | Menu option, the type of ldap search, from the specified root. May take values: <div style="margin-left: 40px;"> subtree
 onelevel
 base </div> |
| 'security' | Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilities. <div style="margin-left: 40px;"> none
 ssl
 sasl </div> |

4.7.2 Function ldaplist

```
(slist) ldaplist(uri,dn,filter,name,scope,security)
```

This function retrieves a single field from all matching LDAP records identified by the search parameters.

ARGUMENTS:

'uri'	String value of the ldap server. e.g. "ldap://ldap.cfengine.com.no"
'dn'	Distinguished name, an ldap formatted name built from components, e.g. "dc=cfengine,dc=com".
'filter'	String filter criterion, in ldap search, e.g. "(sn=User)".
'name'	String value, the name of a single record to be retrieved, e.g. uid.
'scope'	Menu option, the type of ldap search, from the specified root. May take values: <div style="margin-left: 40px;"> subtree onelevel base </div>
'security'	Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilities. <div style="margin-left: 40px;"> none ssl sasl </div>

4.7.3 Function ldaparray

```
(class) ldaparray (array,uri,dn,filter,scope,security)
```

This function retrieves an entire record with all elements and populates an associative array with the entries. It returns a class which is true if there was a match for the search and false if nothing was retrieved.

ARGUMENTS:

'array'	String name of the array to populate with the result of the search
'uri'	String value of the ldap server. e.g. "ldap://ldap.cfengine.com.no"
'dn'	Distinguished name, an ldap formatted name built from components, e.g. "dc=cfengine,dc=com".
'filter'	String filter criterion, in ldap search, e.g. "(sn=User)".
'scope'	Menu option, the type of ldap search, from <div style="margin-left: 40px;"> subtree onelevel base </div>

'security' Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilities.

```
none
ssl
sasl
```

4.7.4 Function regldap

```
(class) regldap(uri,dn,filter,name,scope,regex,security)
```

This function retrieves a single field from all matching LDAP records identified by the search parameters and compares it to a regular expression. If there is a match, true is returned else false.

ARGUMENTS:

'uri' String value of the ldap server. e.g. "ldap://ldap.cfengine.com.no"

'dn' Distinguished name, an ldap formatted name built from components, e.g. "dc=cfengine,dc=com".

'filter' String filter criterion, in ldap search, e.g. "(sn=User)".

'name' String value, the name of a single record to be retrieved, e.g. uid.

'scope' Menu option, the type of ldap search, from the specified root. May take values:

```
subtree
onelevel
base
```

'regex' A regular expression string to match to the results of an LDAP search. If any item matches, the result will be true.

'security' Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilities.

```
none
ssl
sasl
```

4.7.5 LDAP function examples

```
# LDAP example function usage
```

```
body common control
{
bundlesequence => { "ldap" };
}
```

```
bundle agent ldapbundle
{
vars:
```

```

# Get the first matching value for "uid"

"value" string => ldapvalue(
    "ldap://ldap.cfengine.com.no",
    "dc=cfengine,dc=com",
    "(sn=User)",
    "uid",
    "subtree",
    "none"
);

# Get a all matching values for "uid"

"list" slist => ldaplist(
    "ldap://ldap.cfengine.com.no",
    "dc=cfengine,dc=com",
    "(sn=User)",
    "uid",
    "subtree",
    "none"
);

classes:

# Get first matching value with all fields

"gotdata" expression => ldaparray(
    "myarray",
    "ldap://ldap.cfengine.com.no",
    "dc=cfengine,dc=com",
    "(sn=User)",
    "subtree",
    "none"
);

# Look for a regex match within the ldap match

"found" expression => regldap(
    "ldap://ldap.cfengine.com.no",
    "dc=cfengine,dc=com",
    "(sn=User)",
    "uid",
    "subtree",
    "jon.*",
    "none");

```

```
reports:

linux::

    "LDAP VALUE $(value) found";
    "LDAP LIST VALUE $(list)";

gotdata::

    "found data $(ldapbundle.myarray[uid])";

found::

    "Matched regex";

}
```

4.7.6 Best practice for LDAP integration

Relying on LDAP or other network data without validation is a potentially dangerous policy. One could destroy a system by assuming that the service will respond with a sensible result. Cfengine does not recommend reliance on any network services in a configuration policy.

The failure to find an item in LDAP does not imply its non-existence. Policies should work opportunistically. If something is found, it may be used. If something is not found, it is difficult to conclude the reason for this.

If users are removed from a system, a best practice would be to add their names to a list of users who should not exist, rather than assuming that the failure of a search for the user implies his/her absence from the system.

5 File Access Control Lists

5.1 Introduction

Access Control Lists (ACL) allow for a more fine-grained access control on file system objects than standard Unix permissions. In spite of the success of the POSIX model's simplicity the functionality is limited.

File permission security is a subtle topic. Users should take care when experimenting with ACLs as the results can often be counter-intuitive. In some cases the functioning of a system can be compromised by changes of access rights.

Not all file systems support ACLs. In Unix systems there is a plethora of different file system type each with different models of ACLs. Be aware that the mount-options for a file-system often affect the ACL semantics.

Note, recall that when adding a user to a group, this will not have any effect until the next time the user logs in on many operating systems.

As Cfengine works across multiple platforms and needs to support ACLs with different APIs, a common ACL syntax has been abstracted to add a layer of portability. This is a specific feature of Cfengine, not of the host systems. A generic syntax ensures that the ACLs that commonly needed ACLs can be coded in a portable fashion. Cfengine Nova's ACL model is translated into native permissions for implementation; cfengine does not interfere with native access mechanisms in any way. The Cfengine ACL syntax is similar to the POSIX ACL syntax, which is supported by BSD, Linux, HP-UX and Solaris.

Cfengine also allows you to specify platform-dependent ACLs. Of course, these ACLs will only work as expected on the given platform, and must therefore be shielded with classes that select the appropriate model within the promise body.

5.2 File ACL example

The form of a Cfengine files promise that uses ACLs is as follows:

```
#
# test_acl.cf
#

body common control
{
bundlesequence => { "acls" };
}

#####

bundle agent acls

{
files:
```



```

    "/media/flash/acl/test_dir"

    acl => template;
}

#####

body acl template

{
acl_method => "overwrite";
acl_type => "posix";
acl_directory_inherit => "parent";

linux|solaris::

    aces => {
        user::rw,
        group::r,
        mask:rw,
        all:r
    };

windows::

    aces => {
        "user::r(wwx),-r:allow",
        "group::+rw:allow",
        "mask:x:allow",
        "all:r"
    };
}

```

5.2.1 Concepts

There are many different implementations of ACLs. For example, the POSIX draft standard, NTFS and NFSv4 are different and incompatible ACL implementations.

As Cfengine cross-platform, these differences should, for the most usual cases, be transparent to the user. However, some distinctions are impossible to make transparent.

We will explore the different concepts of ACL implementations that are critical to understanding how permissions are defined and enforced in the different file systems. As a running example, we will consider NTFS v. 5 ACLs and Linux POSIX ACLs, because the distinction between these ACL implementations is strong.

5.2.2 Entity types

All ACL implementations have three basic entity types: user, group and all. User and group are simply users and groups of the system, where a group may contain multiple users. all is all users of the system, this type is called "other" in POSIX and "Everyone" in NTFS.

5.2.3 Owners

All file system objects have an owner, which by default is the entity that created the object. The owner can always be a user. However, in some file systems, groups can also be owners (e.g. the "Administrators" group in NTFS). In some ACL APIs, like POSIX, permissions can be set for the owner, i.e. the owner has an independent ACL entry.

5.2.4 Changing owner

It is generally not possible for user A to set user B as the owner of a file system object, even if A owns the object. The superuser ("root" in POSIX, "Administrator" in NTFS) can however always set itself as the owner of an object. In POSIX, the superuser may in fact set any user as the owner, but in NTFS it can only take ownership.

5.2.5 Permissions

An entity can be given a set of permissions on a file system object (e.g. read and write). The data structure holding the permissions for one entity is called an "Access Control Entry". As many users and groups may have sets of permissions on a given object, multiple Access Control Entries are combined to an Access Control List, which is associated with the object.

The set of available permissions differ with ACL implementations. For example, the "Take Ownership" permission in NTFS has no equivalent in POSIX. However, for the most common situations, it is possible to get equivalent security properties by mapping a set of permissions in one file system to another set in a second file system.

There are however different rules for the access to the contents of a directory with no access. In POSIX, no sub-objects of a directory with no access can be accessed. However, in NTFS, sub-objects that the entity has access rights to can be accessed, regardless of whether the entity has access rights to the containing directory.

5.2.6 Deny permissions

If no permissions are given to a particular entity, the entity will be denied any access to the object. But in some file systems, like NTFS, it is also possible to explicitly deny permissions to entities. Thus, two types of permissions exist in these systems: allow and deny.

It is generally good practice to design the ACLs to specify who is allowed to do some operations, in contrary to who is not allowed to do some operations, if possible. The reason for this is that describing who is not allowed to do things tend to lead to more complex rules and could therefore more easily lead to mis-configurations and security holes. A good rule is to only define that users should not be able to access a resource in the following two scenarios:

- Denying access to a subset of a group which is allowed access
- Denying a specific permission when a user or a group has full access

If you think about it, this is the same principle that applies to firewall configuration: it is easier to white-list, specify who should have access, than to blacklist, specify who should not have access. In addition, since Cfengine is designed to be cross-platform and some ACL permissions are not available

on all platforms, we should strive to keep the ACLs as simple as possible. This helps us avoid surprises when the ACLs are enforced by different platforms.

5.2.7 Changing permissions

Generally, only the owner may change permissions on a file system object. However, superusers can also indirectly change permissions by taking ownership first. In POSIX, superusers can change permissions without taking ownership. In NTFS, either ownership or a special permission is needed to change permissions.

5.2.8 Effective permissions

Unfortunately, even though two file systems support all the permissions listed in an ACL, the ACL may be interpreted differently. For a given entity and object with ACL, there are two conceptually different ways to interpret which permissions the entity obtains: ACE precedence and cumulative ACL.

For example, let 'alice' be a user of the group 'staff'. There is an ACL on the file 'schedule', giving alice write permission, and the group 'staff' read permission. We will consider two ways to determine the effective permissions of "alice" to "schedule".

Firstly, by taking the most precise match in the ACL, alice will be granted write permission only. This is because an ACE describing alice is more precise than an ACE describing a group alice is member of. However, note that some ACEs may have the same precedence, like two ACEs describing permissions for groups alice is member of. Then, cumulative matching will be done on these ACEs (explained next). This is how POSIX does it.

Secondly, we can take the cumulative permissions, which yields a user permissions from all the ACE entries with his user name, groups he is member of or the ACE entry specifying all users. In this case, alice would get read and write on "schedule". NTFS computes the effective permissions in this way.

5.2.9 Inheritance

Directories have ACLs associated with them, but they also have the ability to inherit an ACL to sub-objects created within them. POSIX calls the former ACL type "access ACL" and the latter "default ACL", and we will use the same terminology.

In some file systems, like NTFS, objects can only inherit the access ACL of the parent directory (or nothing). There are also differences in what is inherited by default. In POSIX, the default ACL is usually unset, which results in no ACL inheritance, while NTFS always inherits the access ACL to sub-objects.

5.3 Cfengine 3 Generic ACL Syntax

The Cfengine 3 ACL syntax is divided into two main parts, a generic and a file system specific. The generic syntax can be used on any file system for which Cfengine supports ACLs, while the file system specific syntax gives access to all the permissions available under the specified file system.

An ACL can contain both generic and file-system specific syntax. However, if the ACL is enforced on a file system different from that which the file system specific syntax is defined for, the file system specific syntax will be ignored. Thus, only the generic syntax is portable. However, there might be cases where the generic syntax of an ACL is interpreted differently on different file systems. The places where this might occur will be explicitly marked as such. Next, we will discuss the generic syntax. Following sections are devoted to describing the file system specific syntax for each file system that Cfengine supports ACLs for.



Note that even though the same ACL is set on two systems with different ACL types, it may be enforced differently because the ACE matching algorithms differ. For instance, as discussed earlier, NTFS uses cumulative matching, while POSIX uses precedence matching. Cfengine cannot alter the matching algorithms, and simulating one or the other by changing ACL definitions is not possible in all cases, and would probably lead to confusion. Thus, if an ACL is to be used on two systems with different ACL types, the user is encouraged to check that any differences in matching algorithms do not lead to mis-configurations.

```
body acl acl-alias:

acl_method      => "overwrite/append";
acl_type        => "posix/ntfs";
acl_directory_inherit => "parent/specify/none";

aces            =>
    "user:uid:mode[:perm_type]", ...,
    "group:gid:mode[:perm_type]", ...,
    "all:mode[:perm_type]"
    ;

inherit_aces     =>
    "user:uid:mode[:perm_type]", ...,
    "group:gid:mode[:perm_type]", ...,
    "all:mode[:perm_type]"
    ;
```

- `acl-alias` is the name of the defined ACL. It can be any identifier containing alphanumeric characters and underscores. We will use this name when referring to the ACL.
- `acl_method` (optional) can be set to either `overwrite` or `append`, and defaults to `overwrite`. If set to `overwrite`, the defined ACL will replace the currently set ACL. All required fields must then be set in the defined ACL (e.g. `all` in POSIX). Setting it to `append` only adds or modifies the ACEs that are defined.
- `acl_type` (optional) specifies the ACL type. This only needs to be specified if ACL implementation specific syntax is used in the ACL (e.g. ACL specific permissions). If the ACL is enforced on a different ACL implementation than the one stated, the ACL implementation specific syntax is ignored.
- `acl_directory_inherit` (optional) specifies if the ACL should be inherited to newly created sub-objects. Only valid if the ACL is set on a directory. `parent` is the default and indicates that the ACL should be inherited to all sub-objects. If set to `none`, the ACL will not be inherited, but the file system specifies a default ACL, which varies with the file system. If set to `specify`, `inherit_aces` defines the inherited ACL. This is only valid for ACL types where the inherited ACL is allowed to be specified independently.
- `aces` is a list of access control entries. It is parsed from left to right, and multiple entries with the same entity-type and id is allowed. This is necessary to specify permissions with different `perm_type` in the same ACE.
- `inherit_aces` (optional) is a list of access control entries that are set on child objects. It is also parsed from left to right and allows multiple entries with same entity-type and id. Only valid if `acl_directory_inherit` is set to `specify`.

- **user** indicates that the line applies to a user specified by the user identifier `uid`. `mode` is the permission mode string.
- **group** indicates that the line applies to a group specified by the group identifier `gid`. `mode` is the permission mode string.
- **all** indicates that the line applies to every user. `mode` is the permission mode string.
- `uid` is a valid user identifier for the system and cannot be empty. However, `uid` can be set to `*` as a synonym for the entity that owns the file system object (e.g. `user:*:r`).
- `gid` is a valid group identifier for the system and cannot be empty. However, in some acl types, `gid` can be set to `*` to indicate a special group (e.g. in POSIX this refers to the file group).
- `mode` is one or more strings `op|perms|(nperms)`; a concatenation of `op`, `perms` and optionally `(nperms)`, see below, separated with commas (e.g. `+rx,-w(s)`). `mode` is parsed from left to right.
- `op` specifies the operation on any existing permissions, if the defined ACE already exists. `op` can be `=`, empty, `+` or `-`. `=` or empty sets the permissions to the ACE as stated, `+` adds and `-` removes the permissions from any existing ACE.
- `nperms` (optional) specifies file system specific (native) permissions. Only valid if `acl_type` is defined. `nperms` will only be enforced if the file object is stored on a file system supporting the acl type set in `acl_type` (e.g. `nperms` will be ignored if `acl_type:ntfs` and the object is stored on a file system not supporting ntfs ACLs). Valid values for `nperms` varies with different ACL types, and is defined in subsequent sections.
- `perm_type` (optional) can be set to either `allow` or `deny`, and defaults to `allow`. `deny` is only valid if `acl_type` is set to an ACL type that support deny permissions. A `deny` ACE will only be enforced if the file object is stored on a file system supporting the acl type set in `acl_type`.

`gperms` (generic permissions) is a concatenation of zero or more of the characters shown in the table below. If left empty, none of the permissions are set.

Flag	Description	Semantics on file	Semantics on directory
r	Read	Read data, permissions, attributes	Read directory contents, permissions, attributes
w	Write	Write data	Create, delete, rename sub-objects
x	Execute	Execute file	Access sub-objects

Note that the `r` permission is not necessary to read an object's permissions and attributes in all file systems (e.g. in POSIX, having `x` on its containing directory is sufficient).

5.3.1 Generic syntax examples

5.4 Supported ACL types

5.4.1 POSIX

Generic syntax mapping

Entity types All entity types in the generic syntax are mapped to the corresponding entity types with the same name in POSIX, except `all` which corresponds to `other` in POSIX.

Permissions



As shown in the table below, `perms` is mapped straightforward from generic to POSIX permission flags.

Generic flag	POSIX flag
<code>r</code>	<code>r</code>
<code>w</code>	<code>w</code>
<code>x</code>	<code>x</code>

Generic to POSIX permission mapping.

Inheritance POSIX supports `acl_directory_inherit:specify`. The `inherit_aces` list is then set as the default ACL in POSIX (see `acl(5)`).

If `acl_directory_inherit:parent` is set, Cfengine copies the access ACL to the default ACL. Thus, newly created objects get the same access ACL as the containing directory.

`acl_directory_inherit:none` corresponds to no POSIX default ACL. This results in that newly created objects get ACEs for owning user, group and other. The permissions are set in accordance with the mode parameter to the creating function and the umask (usually results in 644 for files and 755 for directories).

5.4.2 POSIX-specific ACL syntax

Native permissions The valid values for `nperms` in POSIX are `r`, `w`, and `x`. These are in fact the same as the generic permissions, so specifying them as generic or native gives the same effect.

File owner and group A user-ACE with uid set to `*` indicates file object owner. A group-ACE with gid set to `*` indicates file group.

mask mask can be defined as a normal ACE, as `mask:mode`. mask specifies the maximum permissions that are granted to named users (not owning user), file group and named groups. mask is optional, and will be computed as explained in `acl_calc_mask(3)` if left unspecified.

Required ACEs

POSIX requires existence of an ACE for the file owner, (`user*:mode`), the file group (`group*:mode`), other (`all:mode`) and mask (`mask:mode`). As mentioned, Cfengine automatically creates a mask-ACE, if missing. However, if `method:overwrite`, the user must ensure that the rest of the required entries are defined.

5.4.3 NTFS

NTFS version 5, which was introduced with Windows 2000, supports thirteen so-called special file permissions, if we do not consider the "Synchronize" permission. The synchronize permission is used to synchronize access to a file handle in a multi-threaded application, and is therefore not directly relevant to security. The thirteen relevant permissions are as follows.

- Traverse Folder/Execute File
- List Folder/Read Data
- Read Attributes
- Read Extended Attributes
- Create Files/Write Data
- Create Folders/Append Data
- Write Attributes
- Write Extended Attributes
- Delete Sub-folders And Files

- Delete
- Read Permissions
- Change Permissions
- Take Ownership

More detail on the semantics of these permissions is available in the references. These thirteen special permissions are grouped to create five sets of permissions: Read, Read-and-Execute, Write, Modify and Full Control. In addition we have the List Folder Contents set, which is equivalent to the Read-and-Execute set but is only available to and inherited by directories. The Full Control set is unsurprisingly all the thirteen special permissions. The special permissions are grouped as follows.

Cfengine maps the 'r' permission to the Read set, the 'w' permission to the Write set, and the 'x' permission to the Traverse Folder or Execute File special permission. This has the effect that 'rx' is mapped to the Read-and-Execute set. Be aware that an 'rwx' permission on NTFS is more restrictive than in POSIX ACLs. This is because NTFS does not allow deletion of objects within a directory without a Delete Sub-folders and Files permission on the directory (or a Delete permission on the object itself), while in POSIX ACLs, 'rwx' on the directory is sufficient to delete any file or directory within it (except when the Sticky-bit is set on the directory). To cover the additional permissions available on NTFS, Cfengine allows the following permissions on NTFS file systems.

Additional permissions:

'd'	Delete
'D'	Delete sub-objects
'p'	Change permissions
'o'	Take ownership

We can see that Cfengine maps rwx to the Modify set in NTFS. For a file, 'rwx' is equivalent in POSIX and NTFS semantics. For a directory, 'rwx' in POSIX semantics is equivalent to 'rwxD' in NTFS semantics. In POSIX ACLs, there is no explicit delete permission, but the write and sticky permissions on the containing directory determines if a user has privileges to delete. In POSIX, the owner and root can change permissions, while usually only the root may change the ownership, so there is no direct equivalent to the Change Permission and Take Ownership in POSIX.

Note that the entity type 'Everyone' in NTFS corresponds to 'other' in Cfengine.

In addition, NTFS supports deny permissions. This can for instance be used to denying a user a particular permission that a group membership grants him. It is important to note that the precedence of allow and deny permissions is as follows:

- Explicit Deny
- Explicit Allow
- Inherited Deny
- Inherited Allow

An important point here is that even though a user is denied access in a parent directory and this permission is inherited, but one of the groups he is member of is explicitly allowed access to a file in that directory, he is actually allowed to access the file.

The default owner is normally an individual user who is currently logged on. The only exceptions occur when the user is a member of either the Administrators group or the Domain Admins group

Owners of NTFS objects can allow another user to take ownership by giving that user Take Ownership permission. owners can give other users permission to Change Permissions (WRITE_DAC). In addition, members of the Administrator group can take ownership never possible to give ownership of an object to another USER. But members of Administrative group can give ownership to that group

Change permissions can be done by owner, by default creator of the filesystem object, and special permissions to allow by others (ntfs).

6 Report extensions

The `cf-report` component generates reports in a variety of formats either for direct viewing (in HTML) or for import into other software systems, using XML or CSV output formats. A control body for the reporter looks like this:

```
body reporter control
{
reports => { "performance", "last_seen", "monitor_history" };
build_directory => "/var/cfengine/reports";
report_output => "html";
}
```

6.1 Reports added in Nova

In Cfengine Nova, one may add the following reports to the list above:

```
'compliance'      Compliance reports.

'setuid'           Setuid log report.

'file_changes'     File change report.

'installed_software' Software package summary and patch level.
```

They are available in all of the standard formats, HTML, XML, and CSV or text.

6.1.1 Performance logs

Performance logs show the last measured and average times taken to complete a particular promise from start to finish. The standard deviation of the time is quoted to show the major variations.

- Promises that have anomalous performance, especially long jobs might be worthy of attention if they do not have a predictable execution time.
- The frequency of long jobs can be restricted
- Jobs that are especially long can be backgrounded for improved throughput.



Promises last kept by atlas.cfengine.com

Copy(localhost:/home/mark > /home/backup)	last performed at Sun Apr 19 08:15	completed in 9.9675 mins	Av 9.4598 mins
Copy(localhost:/home/mark/LapTop > /media/disk-1/CfengineBackupDevice)	last performed at Fri Apr 10 15:55	completed in 4.2757 mins	Av 4.2757 mins
Copy(localhost:/home/mark/LapTop > /media/disk/CfengineBackupDevice)	last performed at Sat Jan 31 11:05	completed in 37.0211 mins	Av 37.0211 mins
Copy(localhost:/home/mark/LapTop > /media/disk/USB_MicroVault_Backup)	last performed at Sun Aug 24 15:00	completed in 18.6437 mins	Av 18.6437 mins
Copy(localhost:/home/mark/LapTop/CfengineProjects/Nova-Knowledge > /cfengine)	last performed at Mon Apr 13 07:07	completed in 0.0132 mins	Av 0.0100 mins

6.1.2 Installed software packages

An overview of installed software as viewed through the eyes of the locally installed package manager. These data are collected as a byproduct of the packages promises. You must check for the existence of

at least one package to generate this report. Moreover, the report can only be generated by sufficiently smart package managers or by custom scripts (see the 'packages' promise in the reference manual).

Software versions installed atlas.cfengine.com

Package	Version	Architecture	Manager
3ddiag	0.742-32.87	x86_64	zypper
3ddiag	0.742-32.87	x86_64	zypper
ConsoleKit	0.2.10-60.26.1	x86_64	zypper
ConsoleKit-32bit	0.2.10-60.26.1	x86_64	zypper
ConsoleKit-x11	0.2.10-60.26.1	x86_64	zypper
DirectFB	1.2.3-6.1	x86_64	zypper
DirectFB	1.2.3-6.1	x86_64	zypper
IlmBase	1.0.1-40.56	x86_64	zypper
IlmBase	1.0.1-40.56	x86_64	zypper
IlmBase-32bit	1.0.1-40.47	x86_64	zypper
IlmBase-32bit	1.0.1-40.47	x86_64	zypper
ImageMagick	6.4.3.6-5.3	x86_64	zypper
ImageMagick	6.4.3.6-5.3	x86_64	zypper
LibVNCServer	0.9.1-154.15	x86_64	zypper
LibVNCServer	0.9.1-154.15	x86_64	zypper
Mesa	7.2-10.2.1	x86_64	zypper
Mesa-32bit	7.2-10.2.1	x86_64	zypper

6.1.3 A promise compliance report

Each time `cf-agent` completes a scan of whether local promises are kept it computes a simple high level summary of the state of the system in terms of whether promises were kept before and after execution.

The agent distinguished three categories

Promise kept

The promise was already kept, and no action was required on the system.

Promise repaired

The promise was not properly kept but maintenance was performed and the problem was healed. The system is now in compliance.

Promise not kept

The promise was not kept, and no remedial action was taken to correct it. (This is usually a result of an explicit policy decision to not repair a problem, e.g. selecting action 'warn' instead of 'fix'.)



Policy compliance summary on atlas.cfengine.com

Start check	End check	Policy version	% scheduled promises kept	% scheduled promises repaired	% scheduled promises ignored
Sun Apr 19 12:45:02 2009	Sun Apr 19 12:45:04 2009:	(not specified) (agent-1)	88%	12%	0%
Sun Apr 19 12:45:02 2009	Sun Apr 19 12:45:02 2009:	(not specified) (agent-0)	100%	0%	0%
Sun Apr 19 12:35:02 2009	Sun Apr 19 12:35:04 2009:	(not specified) (agent-1)	88%	12%	0%
Sun Apr 19 12:35:02 2009	Sun Apr 19 12:35:02 2009:	(not specified) (agent-0)	100%	0%	0%
Sun Apr 19 12:25:02 2009	Sun Apr 19 12:25:03 2009:	(not specified) (agent-1)	88%	12%	0%
Sun Apr 19 12:25:01 2009	Sun Apr 19 12:25:02 2009:	(not specified) (agent-0)	100%	0%	0%
Sun Apr 19 12:15:03 2009	Sun Apr 19 12:15:04 2009:	(not specified) (agent-1)	88%	12%	0%
Sun Apr 19 12:15:02 2009	Sun Apr 19 12:15:03 2009:	(not specified) (agent-0)	98%	2%	0%
Sun Apr 19 12:06:07 2009	Sun Apr 19 12:06:08 2009:	(not specified) (agent-0)	98%	2%	0%
Sun Apr 19 12:01:30 2009	Sun Apr 19 12:05:56 2009:	(not specified) (agent-0)	79%	21%	0%

6.1.4 A file content change report

Hash scans of files are a common way of detecting any small changes in file integrity. This log shows the specific times that change events were detected. The resolution of this log depends on the frequency of scanning, which is the purview of a `files` promise.

File hash-change events recorded on atlas.cfengine.com

Time of change event	Filename
Sun Apr 19 12:04:57 2009	/usr/bin/pg_controldata
Sun Apr 19 12:04:56 2009	/usr/bin/psql
Sun Apr 19 12:04:54 2009	/usr/bin/gtk-query-immodules-2.0
Sun Apr 19 12:04:53 2009	/usr/bin/kbluetooth4
Sun Apr 19 12:04:52 2009	/usr/bin/flashplayer
Sun Apr 19 12:04:50 2009	/usr/bin/nm-tool
Sun Apr 19 12:04:48 2009	/usr/bin/kdm
Sun Apr 19 12:04:44 2009	/usr/bin/hcitool
Sun Apr 19 12:04:42 2009	/usr/bin/pg_dump
Sun Apr 19 12:04:40 2009	/usr/bin/last
Sun Apr 19 12:04:40 2009	/usr/bin/solid-bluetooth
Sun Apr 19 12:04:36 2009	/usr/bin/jpegicc
Sun Apr 19 12:04:29 2009	/usr/bin/pg_restore
Sun Apr 19 12:04:24 2009	/usr/bin/reindexdb
Sun Apr 19 12:04:21 2009	/usr/bin/dfutool
Sun Apr 19 12:04:15 2009	/usr/bin/sdptool
Sun Apr 19 12:04:14 2009	/usr/bin/ioport

6.1.5 Installed setuid program report

Setuid root programs (Unix systems) are programs given special privileges to carry out actions normally restricted to Administrative or root users. Attention to programs with setuid permissions is a matter

of security, and thus cfengine watches out for changes in the known list of these programs as a matter of course. This report summarizes the known programs.

Known setuid programs on atlas.cfengine.com

Filename
/usr/bin/Xorg
/usr/bin/at
/usr/bin/chage
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/crontab
/usr/bin/expiry
/usr/bin/fileshareset
/usr/bin/fusermount
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/rcp
/usr/bin/rlogin

7 Server extensions

Cfengine Nova adds a simple server extension to the Community Edition server, namely the ability to encode data directly in policy. This feature is useful for distributing password hashes to systems.

7.1 Server access resource type

By default, access to resources granted by the server are files. However, sometimes it is useful to cache `literal` strings, hints and data in the server, e.g. the contents of variables, hashed passwords etc for easy access. In the case of literal data, the promise handle serves as the reference identifier for queries. Queries are instigated by function calls by any agent.

`access:`

```
"This is a string with a $(localvar) for remote collection"

    handle  => "test_scalar",
    resource_type => "literal",
    admit   => { "127.0.0.1" };
```

The promise looks exactly like a promise for a file object, but the data are literal and entered into the policy. This is a useful approach for distributing password hashes on a need-to-know basis from a central location. The server configuration file need not be distributed to any client, thus only authorized systems will have access to the hashes.

7.2 Function remotescalar

The client side of the literal look up function is:

```
(string) remotescalar(resource handle, host/IP address, encrypt);
```

This function downloads a string from a remote server, using the promise handle as a variable identifier.

ARGUMENTS:

`'resource handle'`

The name of the promise on the server side

`'host or IP address'`

The location of the server on which the resource resides.

`'encrypt'`

Whether to encrypt the connection to the server.

```
true
yes
false
no
```

7.3 Example remote scalar lookup

```
#####
#
# Remote value from server connection to cfServer
#
#####

body common control

{
  bundlesequence => { "testbundle" };

  version => "1.2.3";
}

#####

bundle agent testbundle

{
  vars:

    "remote" string => remotescalar("test_scalar","127.0.0.1","yes");

  reports:

    linux::

      "Receive value ${remote}";
}

#####
# Server config
#####

body server control

{
  allowconnects      => { "127.0.0.1" , "::1" };
  allowallconnects   => { "127.0.0.1" , "::1" };
  trustkeysfrom      => { "127.0.0.1" , "::1" };
  allowusers         => { "mark" };
}

#####
```



```
bundle server access_rules()

{
  vars:

    "localvar" string => "literal string";

  access:

    "This is a $(localvar) for remote access"

    handle  => "test_scalar",
    resource_type => "literal",
    admit    => { "127.0.0.1" };
}
```


Table of Contents

1	Introduction to Nova.....	1
1.1	Installing Cfengine Nova	1
1.1.1	Building cfengine yourself.....	1
1.2	Installing from packages	2
1.3	Enhancements	2
2	Productivity and Documentation.....	5
2.1	Syntax lookup on the command line.....	5
2.2	Knowledge map creation	5
2.2.1	Procedure for generating a local knowledge map.....	8
2.2.2	The 'knowledge.cf' file	10
2.2.3	Trouble shooting the knowledge base	10
3	Monitoring extensions.....	13
3.1	Long term trends	13
3.2	Custom promises to measure	13
3.2.1	Extraction strings and logging	13
3.2.2	Extracting one-off numerical data	14
3.2.3	Extraction to list variable.....	15
3.3	Uses for custom monitoring	16
4	Databases.....	17
4.1	How to manage databases.....	17
4.2	Database access rights	17
4.3	Creating a point of contact on a server	17
4.4	Creating SQL databases.....	18
4.4.1	Creating a database manually	18
4.4.2	Creating a database directly	19
4.5	Database table promises.....	20
4.6	MS Registry functions.....	21
4.6.1	Creating a registry key	21
4.6.2	Creating a value-data pair	22
4.6.3	Scanning and restoring the registry	23
4.7	LDAP integration.....	24
4.7.1	Function ldapvalue.....	24
4.7.2	Function ldaplist	25
4.7.3	Function ldaparray.....	25
4.7.4	Function regldap	26
4.7.5	LDAP function examples.....	26
4.7.6	Best practice for LDAP integration	28

5	File Access Control Lists	29
5.1	Introduction	29
5.2	File ACL example	29
5.2.1	Concepts	30
5.2.2	Entity types	31
5.2.3	Owners	31
5.2.4	Changing owner	31
5.2.5	Permissions	31
5.2.6	Deny permissions	31
5.2.7	Changing permissions	32
5.2.8	Effective permissions	32
5.2.9	Inheritance	32
5.3	Cfengine 3 Generic ACL Syntax	32
5.3.1	Generic syntax examples	34
5.4	Supported ACL types	34
5.4.1	POSIX	34
5.4.2	POSIX-specific ACL syntax	35
5.4.3	NTFS	35
6	Report extensions	39
6.1	Reports added in Nova	39
6.1.1	Performance logs	39
6.1.2	Installed software packages	40
6.1.3	A promise compliance report	41
6.1.4	A file content change report	43
6.1.5	Installed setuid program report	43
7	Server extensions	45
7.1	Server access resource type	45
7.2	Function <code>remotescalar</code>	45
7.3	Example remote scalar lookup	46