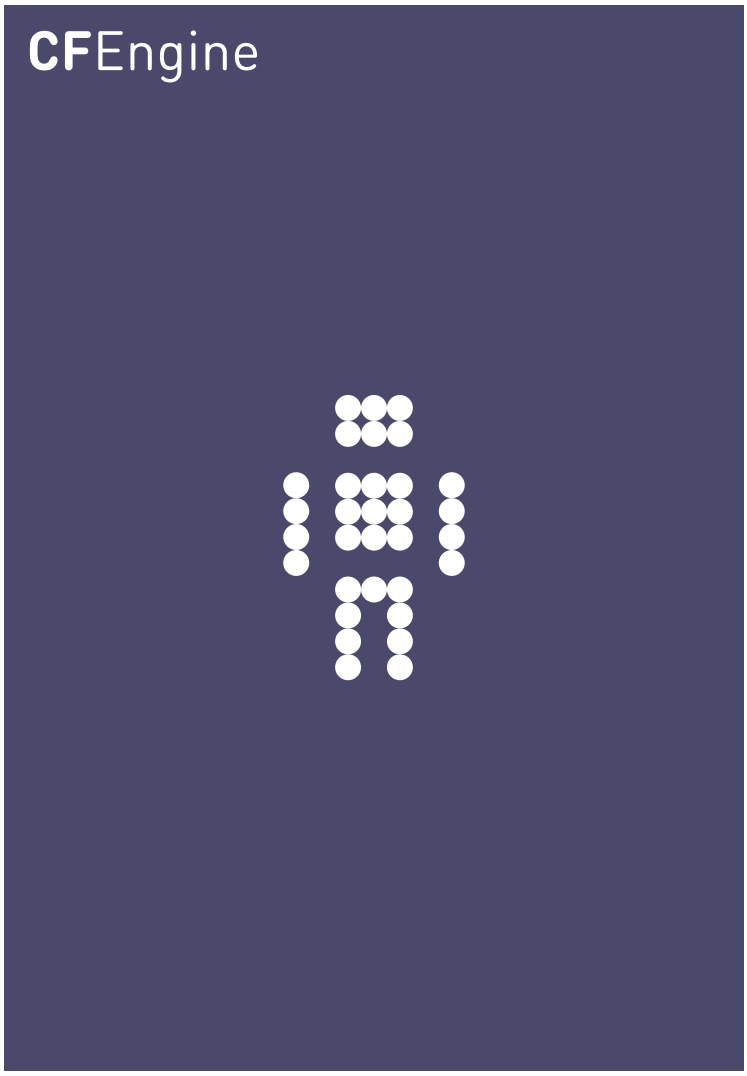


Windows Management with CFEngine Enterprise

A CFEngine Special Topics Handbook

CFEngine AS

CFEngine Enterprise is a cross-platform and versatile tool that unifies the desired state management on all major operating systems.

However, some operating systems are designed fundamentally different than others, requiring special CFEngine language features when being described. In this document, we highlight the CFE Enterprise extensions provided to the Windows platform.

Table of Contents

System requirements

CFEngine Enterprise, being so lightweight, works equally well on Windows clients as on Windows servers. Both native 32-bit/x86 (package name `i686`) and 64-bit/x64 (package name `x86_64`) packages are available to customers. It is important that you select the `x86_64` package if you are running 64-bit Windows.

Of Windows client operating systems, anything from Windows XP SP2 and newer is supported. On the server side, Windows Server 2003 and newer is supported.

CFEngine Enterprise communicates bi-directionally on port `5308`, so make sure that this port is open for outgoing and incoming `TCP` connections.

All software dependencies are bundled with the CFEngine Enterprise package. The total disk consumption is about 70 `MB`, and the memory usage is less than 30 `MB`.

Installation

The installation and set-up procedure on Windows is not different than that for other operating systems CFE Enterprise runs on, so the CFE Enterprise getting started document available at http://software.cfengine.com applies to the Windows version as well.

The Windows `msi`-packages will get silently installed (no prompts) to `Cfengine` under your program files directory (e.g. `C:\Program Files\Cfengine` on English Windows versions). It is important that the installer is run with Administrative priviliges. To ensure this, open a `Command Prompt` in Administrative mode and run '`msiexec -i cfengine-nova-VERSION-ARCH.msi`' (replace `VERSION` and `ARCH` appropriately).

If you are just going to test your policies on a Windows host, it is more efficient to not bootstrap to a policy server, but run the policies locally just after you create them. You can install the license with the `cf-key -l` command – you will need to copy over the licensed public key as advised by `cf-key -l`.[1]

Eventually, when you are done testing and want to bootstrap a Windows host to a policy server, please run the following command (against a Linux-based policy server, as advised in the CFE Enterprise getting started document). If we assume the policy server's IP address is '123.456.789.123', you need to run the following command to bootstrap the Windows host.

```
C:\Program Files\Cfengine\bin\cf-agent.exe --bootstrap --policy-server 123.456.789.123
```

Testing policies locally

Create a new text file `Cfengine\inputs\promises.cf` and input the following text using your favourite text editor.

```
body common control
{
bundlesequence => { "test" };
inputs => { "cfengine_stdlib.cf" };
host_licenses_paid => "1";
```

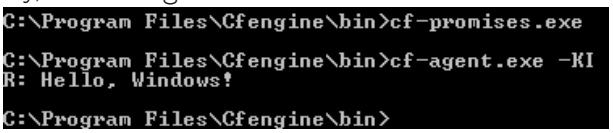
[1] Avaliable in CFEngine Enterprise 3.0 and beyond. Run `cf-key -l C:\path\to\license.dat` and follow the instructions.

```
}

bundle agent test
{
reports:
windows::
  "Hello, Windows!";
}
```

Now, go to your terminal (e.g. Command Prompt) and navigate to `Cfengine\bin` under program files. Run `cf-promises.exe`. It should generate no output, which indicates correct syntax and license.

To execute the policy, run `cf-agent.exe -K`. You should see the following output.



We now have a basic skeleton policy that we can test our Windows promises with. These can later be integrated at the policy hub to ensure that they are run on all Windows systems. We will assume this general skeleton for the rest of this document, modifying the contents of the `test` bundle only.

Windows registry management

CFEngine Enterprise supports fine-grained management of the Windows registry. These promises are encapsulated under the `databases:` promise type.

Creating values

Let us modify our skeleton bundle to contain the following.

```
...
bundle agent test
{
databases:

  "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security"

    database_operation => "create",
    database_rows => { "MaxSize,REG_DWORD,84017152", "Retention,REG_DWORD,0"},
    database_type => "ms_registry",
    comment       => "Ensure eventlog size is in accordance with standards";
}
```

Now, we again run `cf-promises.exe` to ensure the syntax is correct, followed by `cf-agent.exe -KI`. Note that we added the `-I` option which tells `cf-agent.exe` to notify us on the existing state of the system and any actions done to ensure the desired state. The output should look like the following.

```
C:\Program Files\Cfengine\bin>cf-promises.exe

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
 -> Repairing registry REG_DWORD value as (MaxSize,84017152)
 -> Verified value (Retention,0) correct for HKEY_LOCAL_MACHINE\SYSTEM\CurrentCo
ntrolSet\Services\Eventlog\Security

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
 -> Verified value (MaxSize,84017152) correct for HKEY_LOCAL_MACHINE\SYSTEM\Curr
entControlSet\Services\Eventlog\Security
 -> Verified value (Retention,0) correct for HKEY_LOCAL_MACHINE\SYSTEM\CurrentCo
ntrolSet\Services\Eventlog\Security

C:\Program Files\Cfengine\bin>
```

When we run `cf-agent.exe` twice, the second run will do nothing because the first run has already corrected the value. This is convergence — CFEngine is ensuring the desired state.

Removing values

In order to remove values instead, we just need to adjust the policy slightly, resulting in the following bundle.

```
...
bundle agent test
{
databases:

  "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security"

    database_operation => "delete",
    database_columns   => { "value1", "value2"},
    database_type      => "ms_registry",
    comment            => "Remove stray values generated by an application";
}
```

Now, if you create 'value1' and 'value2' in the key above, `cf-agent.exe` should show the following output.

```
C:\Program Files\Cfengine\bin>cf-promises.exe

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
 -> Registry value "value1" in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Servi
ces\Eventlog\Security deleted...
 -> Registry value "value2" in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Servi
ces\Eventlog\Security deleted...

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
 -> Registry value "value1" in "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Serv
ices\Eventlog\Security" already gone
 -> Registry value "value2" in "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Serv
ices\Eventlog\Security" already gone
```

At the time of writing, CFE Enterprise supports the `REG_DWORD` (double word), `REG_SZ` (string) and `REG_EXPAND_SZ` (expandable string) data types, as given in the middle field of the `database_rows` list elements. See the CFEngine reference manual for an updated list of supported data types.

Also note the `registryvalue()` function which can be used to read out value data from the registry and act upon it. Examples of its use are also available in the CFEngine reference manual.

Windows service management

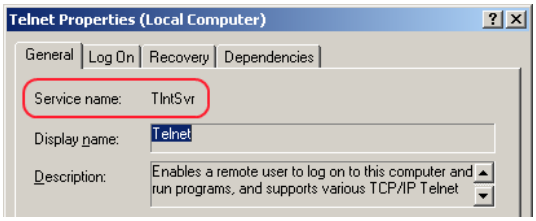
CFEngine Enterprise can maintain complete control of the state of all Windows services. For example, services prone to security issues or errors can easily be given a disabled state.

```
services:

  "TlntSvr"
     service_policy => "disable";
```

A service can also be given a running state, in which case CFEngine Enterprise ensures that it is running, and starts it if it is not, with parameters if desired. More advanced policy options are also available, including support for starting and stopping dependencies, and configuring when the services should be started (e.g. only when they are being used).

Note that the name of a service in Windows may be different from its "Display name". CFEngine Enterprise policies use the name, not the display name, due to the need of uniqueness.



A complete example of a service management bundle is show below.

```
...
bundle agent test
{
services:

windows::
 "W32Time"
     service_policy => "start",
     service_method => bootstart,
     comment => "Ensure important services are running and starting at boot";

Windows_Server_2008::
  "RemoteRegistry"
     service_policy => "disable",
     service_method => force_deps,
     comment => "Disable services that create security issues";

}
```

This example ensures that the Windows Time service is running on all Windows hosts, and that Remote registry is disabled on all Windows 2008 servers.

File and folder permissions

CFEngine Enterprise can ensure the permissions or Access Control Lists (ACLs) of your Windows systems are correctly set. Windows ACLs are a complex topic by itself, with support for more than ten different permission bits and inheritance. CFE Enterprise supports all of this, but we will just cover the basics in this document.

The following policy will ensure strict permissions on a directory 'C:\Secret' and a file 'C:\Secret\file.txt'.

```
...

bundle agent test
{
vars:
  "acl_secret_dir" slist => { "user:Administrator:rwx:allow",
                              "group:Administrators:rx:allow" };
  "acl_secret_file" slist => { "user:Administrator:rw:allow" };

files:

windows::
  "C:\Secret",
    acl => ntfs( "@(acl_secret_dir)" ),
        depth_search => include_base,
        perms => owner( "Administrator" );

  "C:\Secret\file.txt",
    acl => ntfs( "@(acl_secret_file)" ),
        perms => owner( "Administrator" );
}
```

The CFEngine reference manual contains a description of all the available ACL options. Also refer to the the CFEngine Enterprise Owner's manual for a more in-depth discussion of the ACL options available.

Windows-aware features in CFE Enterprise

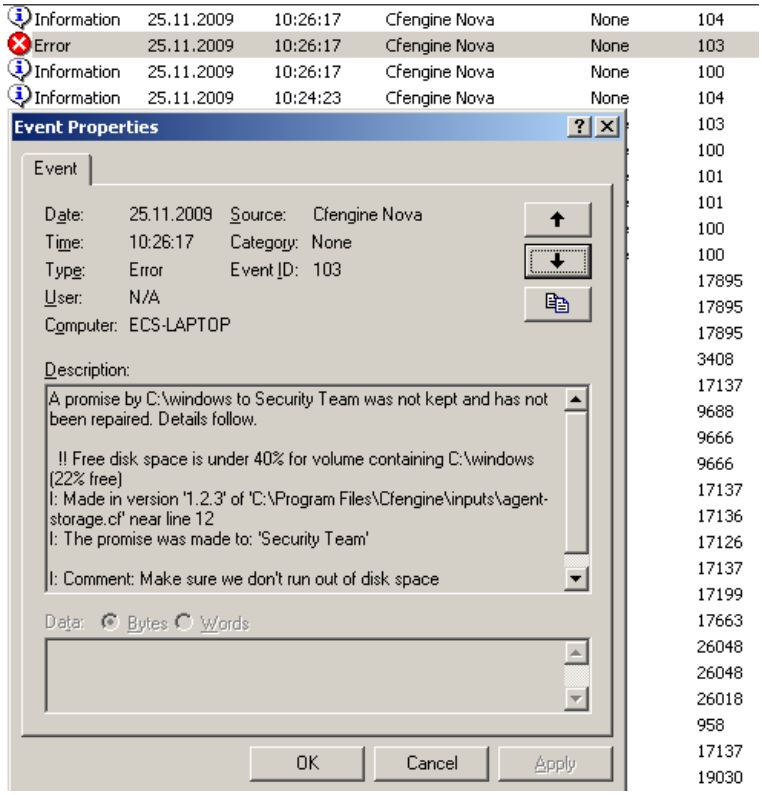
CFEngine Enterprise integrates with the Windows operating system in multiple ways.

The CFEngine scheduler in CFE Enterprise (cf-execd) runs as a Windows service. This means it runs in the background and starts with Windows, before any user logs in. It can be configured, started and stopped from the "Services" listing in Windows, just like any other Windows service.

Event logs are the Windows counterpart to syslog from Unix. The main difference is that event logs aim to group similar log messages, giving each group an event id. The following event ids are defined in CFEngine Enterprise, allowing categorisation of the log message based on its type. The CFE Enterprise event logs can be found under the "System" logs.

Description	**Event ID**	**Type**
Promise kept	100	Information
Promise repaired	101	Information
Promise not repaired due warn only policy	102	Error
Promise not repaired due to error	103	Error
Report promise	104	Information

Generic information	105	Information
Generic verbose	106	Information
Generic warning	107	Warning
Generic error	108	Error



By default, only promise not repaired and generic error events are logged to avoid flooding the Event Log. You can turn on verbose logging to log all messages, like the following example.

```
body common control
{
inputs => { "cfengine_stdlib.cf" };
bundlesequence  => { "main" };
}

bundle agent main
{
files:
"C:\test.txt"
  create => "true",
  action => log_verbose;
}
```

Windows special variables

Three new special variables have been added to the Windows version of CFEngine Enterprise.

- `sys.windir` contains the Windows directory, e.g. "C:\WINDOWS".
- `sys.winsysdir` contains the Windows system directory, e.g. "C:\WINDOWS\system32".
- `sys.winprogdir` contains the program files directory, e.g. "C:\Program Files".

Note that these variables are not statically coded, but retrieved from the current system. For example, `sys.winprogdir` is often different on Windows versions in distinct languages.

Windows hard classes

The Windows version of CFEngine Enterprise defines hard classes to pinpoint the exact version of Windows that it is running on, the service pack version and if it's a server or workstation.

First of all, the class `windows` is defined on all Windows platforms. For Windows workstations, such as Windows XP, `WinWorkstation` is defined. On Windows servers, such as Windows Server 2003, `WinServer` is defined. In addition, if the server is a domain controller, `DomainController` is defined. Note that if `DomainController` is defined, then `WinServer` is also defined, for natural reasons.

The class `Service_Pack_X_Y` is defined according to the service pack version. For example, at the time of writing, `Service_Pack_3_0` is set on an updated Windows XP operating system.

To allow taking specific actions on different Windows versions, one of the following hard classes is defined.

- `Windows_7`
- `Windows_Server_2008_R2`
- `Windows_Server_2008`
- `Windows_Vista`
- `Windows_Server_2003_R2`
- `Windows_Home_Server`
- `Windows_Server_2003`
- `Windows_XP_Professional_x64_Edition`
- `Windows_XP`
- `Windows_2000`

Note that all defined hard classes for a given system is shown by running `cf-promises -v`.

Notes on windows policies

A potential problem source when writing policies for windows is that paths to executables often contain spaces. This makes it impossible for CFEngine to know where the executable ends and the parameters to it starts. To solve this, we place escaped quotes around the executable. Windows share paths (double backslashes) also need escaping.

Additionally, Windows does not support that processes start themselves in in the background (i.e. fork off a child process in the Unix world). The result is that CFEngine is always waiting for the commands to finish execution before checking the next promise. To avoid this, use the background attribute in the action body-part.

Both these things are demonstrated in the following example.

```
body common control
{
inputs => { "cfengine_stdlib.cf" };
bundlesequence  => { "main" };
}

bundle agent main
{
commands:

"\"C:\Program Files\Some Dir\program name.bat\" --silent --batch"
  action => in_shell_bg;

"\"\\\\computer\share path\my program.exe\" /some args";
}
```

Finally, one should note that Windows lacks support for certain features that are utilised in Unix versions of CFEngine. These include symbolic links, file groups, user and group identifiers.

Thus, the parts of promises containing these features will be ignored. For example, the `getgid()` function does not return anything on Windows. The CFEngine reference manual documents exactly which promises are ignored and not. Also, `cf-agent.exe` from CFEngine Enterprise prints warning messages on ignored attributes when run in verbose mode (`cf-agent.exe -Kv`).