NAME

config − OpenSSL CONF library configuration files

DESCRIPTION

The OpenSSL CONF library can be used to read configuration files. It is used for the OpenSSL master configuration file **openssl.cnf** and in a few other places like **SPKAC** files and certificate extension files for the **x509** utility. OpenSSL applications can also use the CONF library for their own purposes.

A configuration file is divided into a number of sections. Each section starts with a line **[section_name]** and ends when a new section is started or end of file is reached. A section name can consist of alphanumeric characters and underscores.

The first section of a configuration file is special and is referred to as the **default** section. This section is usually unnamed and spans from the start of file until the first named section. When a name is being looked up it is first looked up in a named section (if any) and then the default section.

The environment is mapped onto a section called **ENV**.

Comments can be included by preceding them with the **#** character

Other files can be included using the **.include** directive followed by a path. If the path points to a directory all files with names ending with **.cnf** or **.conf** are included from the directory. Recursive inclusion of directories from files in such directory is not supported. That means the files in the included directory can also contain **.include** directives but only inclusion of regular files is supported there. The inclusion of directories is not supported on systems without POSIX IO support.

It is strongly recommended to use absolute paths with the **.include** directive. Relative paths are evaluated based on the application current working directory so unless the configuration file containing the **.include** directive is application specific the inclusion will not work as expected.

There can be optional **=** character and whitespace characters between **.include** directive and the path which can be useful in cases the configuration file needs to be loaded by old OpenSSL versions which do not support the **.include** syntax. They would bail out with error if the **=** character is not present but with it they just ignore the include.

Each section in a configuration file consists of a number of name and value pairs of the form **name=value**

The **name** string can contain any alphanumeric characters as well as a few punctuation symbols such as **. , ;** and **_**.

The **value** string consists of the string following the **=** character until end of line with any leading and trailing white space removed.

The value string undergoes variable expansion. This can be done by including the form **$var** or **${var}**: this will substitute the value of the named variable in the current section. It is also possible to substitute a value from another section using the syntax **$section::name** or **${section::name}**. By using the form **$ENV::name** environment variables can be substituted. It is also possible to assign values to environment variables by using the name **ENV::name**, this will work if the program looks up environment variables using the **CONF** library instead of calling **getenv()** directly. The value string must not exceed 64k in length after variable expansion. Otherwise an error will occur.

It is possible to escape certain characters by using any kind of quote or the \ character. By making the last character of a line a \ a **value** string can be spread across multiple lines. In addition the sequences **\n**, **\r**, **\b** and **\t** are recognized.

All expansion and escape rules as described above that apply to **value** also apply to the path of the **.include** directive.

OPENSSL LIBRARY CONFIGURATION

Applications can automatically configure certain aspects of OpenSSL using the master OpenSSL configuration file, or optionally an alternative configuration file. The **openssl** utility includes this functionality: any sub command uses the master OpenSSL configuration file unless an option is used in the sub command to use an alternative configuration file.

To enable library configuration the default section needs to contain an appropriate line which points to the main configuration section. The default name is **openssl_conf** which is used by the **openssl** utility. Other applications may use an alternative name such as **myapplication_conf**. All library configuration lines appear in the default section at the start of the configuration file.

The configuration section should consist of a set of name value pairs which contain specific module configuration information. The **name** represents the name of the *configuration module*. The meaning of the **value** is module specific: it may, for example, represent a further configuration section containing configuration module specific information. E.g.:

```
# This must be in the default section
openssl_conf = openssl_init

[openssl_init]

oid_section = new_oids
engines = engine_section

[new_oids]

... new oids here ...

[engine_section]

... engine stuff here ...
```

The features of each configuration module are described below.

ASN1 Object Configuration Module

This module has the name **oid_section**. The value of this variable points to a section containing name value pairs of OIDs: the name is the OID short and long name, the value is the numerical form of the OID. Although some of the **openssl** utility sub commands already have their own ASN1 OBJECT section functionality not all do. By using the ASN1 OBJECT configuration module **all** the **openssl** utility sub commands can see the new objects as well as any compliant applications. For example:

```
[new_oids]

some_new_oid = 1.2.3.4
some_other_oid = 1.2.3.5
```

It is also possible to set the value to the long name followed by a comma and the numerical OID form. For example:

```
shortName = some object long name, 1.2.3.4
```

Engine Configuration Module

This ENGINE configuration module has the name **engines**. The value of this variable points to a section containing further ENGINE configuration information.

The section pointed to by **engines** is a table of engine names (though see **engine_id** below) and further sections containing configuration information specific to each ENGINE.

Each ENGINE specific section is used to set default algorithms, load dynamic, perform initialization and send ctrls. The actual operation performed depends on the *command* name which is the name of the name value pair. The currently supported commands are listed below.

For example:

```
[engine_section]

# Configure ENGINE named "foo"
foo = foo_section
```

```
# Configure ENGINE named "bar"
bar = bar_section

[foo_section]
... foo ENGINE specific commands ...

[bar_section]
... "bar" ENGINE specific commands ...
```

The command **engine_id** is used to give the ENGINE name. If used this command must be first. For example:

```
[engine_section]
# This would normally handle an ENGINE named "foo"
foo = foo_section

[foo_section]
# Override default name and use "myfoo" instead.
engine_id = myfoo
```

The command **dynamic_path** loads and adds an ENGINE from the given path. It is equivalent to sending the ctrls **SO_PATH** with the path argument followed by **LIST_ADD** with value 2 and **LOAD** to the dynamic ENGINE. If this is not the required behaviour then alternative ctrls can be sent directly to the dynamic ENGINE using ctrl commands.

The command **init** determines whether to initialize the ENGINE. If the value is **0** the ENGINE will not be initialized, if **1** and attempt it made to initialized the ENGINE immediately. If the **init** command is not present then an attempt will be made to initialize the ENGINE after all commands in its section have been processed.

The command **default_algorithms** sets the default algorithms an ENGINE will supply using the functions **ENGINE_set_default_string()**.

If the name matches none of the above command names it is assumed to be a ctrl command which is sent to the ENGINE. The value of the command is the argument to the ctrl command. If the value is the string **EMPTY** then no value is sent to the command.

For example:

```
[engine_section]

# Configure ENGINE named "foo"
foo = foo_section

[foo_section]
# Load engine from DSO
dynamic_path = /some/path/fooengine.so
# A foo specific ctrl.
some_ctrl = some_value
# Another ctrl that doesn't take a value.
other_ctrl = EMPTY
# Supply all default algorithms
default_algorithms = ALL
```

EVP Configuration Module

This modules has the name **alg_section** which points to a section containing algorithm commands.

Currently the only algorithm command supported is **fips_mode** whose value can only be the boolean string **off**. If **fips_mode** is set to **on**, an error occurs as this library version is not FIPS capable.

SSL Configuration Module

This module has the name **ssl_conf** which points to a section containing SSL configurations.

Each line in the SSL configuration section contains the name of the configuration and the section containing it.

Each configuration section consists of command value pairs for **SSL_CONF**. Each pair will be passed to a **SSL_CTX** or **SSL** structure if it calls **SSL_CTX_config()** or **SSL_config()** with the appropriate configuration name.

Note: any characters before an initial dot in the configuration section are ignored so the same command can be used multiple times.

For example:

```
ssl_conf = ssl_sect

[ssl_sect]

server = server_section

[server_section]

RSA.Certificate = server-rsa.pem
ECDSA.Certificate = server-ecdsa.pem
Ciphers = ALL:!RC4
```

The system default configuration with name **system_default** if present will be applied during any creation of the **SSL_CTX** structure.

Example of a configuration with the system default:

```
ssl_conf = ssl_sect

[ssl_sect]
system_default = system_default_sect

[system_default_sect]
MinProtocol = TLSv1.2
MinProtocol = DTLSv1.2
```

NOTES

If a configuration file attempts to expand a variable that doesn't exist then an error is flagged and the file will not load. This can happen if an attempt is made to expand an environment variable that doesn't exist. For example in a previous version of OpenSSL the default OpenSSL master configuration file used the value of **HOME** which may not be defined on non Unix systems and would cause an error.

This can be worked around by including a **default** section to provide a default value: then if the environment lookup fails the default value will be used instead. For this to work properly the default value must be defined earlier in the configuration file than the expansion. See the **EXAMPLES** section for an example of how to do this.

If the same variable exists in the same section then all but the last value will be silently ignored. In certain circumstances such as with DNs the same field may occur multiple times. This is usually worked around by ignoring any characters before an initial `.` e.g.

```
1.OU="My first OU"
2.OU="My Second OU"
```

EXAMPLES

Here is a sample configuration file using some of the features mentioned above.

```
# This is the default section.

HOME=/temp
RANDFILE= ${ENV::HOME}/.rnd
configdir=$ENV::HOME/config

[ section_one ]

# We are now in section one.

# Quotes permit leading and trailing whitespace
any = " any variable name "

other = A string that can \
cover several lines \
by including \\ characters

message = Hello World\n

[ section_two ]

greeting = $section_one::message
```

This next example shows how to expand environment variables safely.

Suppose you want a variable called **tmpfile** to refer to a temporary filename. The directory it is placed in can determined by the **TEMP** or **TMP** environment variables but they may not be set to any value at all. If you just include the environment variable names and the variable doesn't exist then this will cause an error when an attempt is made to load the configuration file. By making use of the default section both values can be looked up with **TEMP** taking priority and /**tmp** used if neither is defined:

```
TMP=/tmp
# The above value is used if TMP isn't in the environment
TEMP=$ENV::TMP
# The above value is used if TEMP isn't in the environment
tmpfile=${ENV::TEMP}/tmp.filename
```

Simple OpenSSL library configuration example to enter FIPS mode:

```
# Default appname: should match "appname" parameter (if any)
# supplied to CONF_modules_load_file et al.
openssl_conf = openssl_conf_section

[openssl_conf_section]
# Configuration module list
alg_section = evp_sect

[evp_sect]
# Set to "yes" to enter FIPS mode if supported
fips_mode = yes
```

Note: in the above example you will get an error in non FIPS capable versions of OpenSSL.

Simple OpenSSL library configuration to make TLS 1.2 and DTLS 1.2 the system-default minimum TLS and DTLS versions, respectively:

```
# Toplevel section for openssl (including libssl)
openssl_conf = default_conf_section
```

```
[default_conf_section]
# We only specify configuration for the "ssl module"
ssl_conf = ssl_section

[ssl_section]
system_default = system_default_section

[system_default_section]
MinProtocol = TLSv1.2
MinProtocol = DTLSv1.2
```

The minimum TLS protocol is applied to **SSL_CTX** objects that are TLS-based, and the minimum DTLS protocol to those are DTLS-based. The same applies also to maximum versions set with **MaxProtocol**.

More complex OpenSSL library configuration. Add OID and don't enter FIPS mode:

```
# Default appname: should match "appname" parameter (if any)
# supplied to CONF_modules_load_file et al.
openssl_conf = openssl_conf_section

[openssl_conf_section]
# Configuration module list
alg_section = evp_sect
oid_section = new_oids

[evp_sect]
# This will have no effect as FIPS mode is off by default.
# Set to "yes" to enter FIPS mode, if supported
fips_mode = no

[new_oids]
# New OID, just short name
newoid1 = 1.2.3.4.1
# New OID shortname and long name
newoid2 = New OID 2 long name, 1.2.3.4.2
```

The above examples can be used with any application supporting library configuration if ‘‘openssl_conf’’ is modified to match the appropriate ‘‘appname’’.

For example if the second sample file above is saved to ‘‘example.cnf’’ then the command line:

```
OPENSSL_CONF=example.cnf openssl asn1parse −genstr OID:1.2.3.4.1
```

will output:

```
    0:d=0  hl=2 l=   4 prim: OBJECT            :newoid1
```

showing that the OID ‘‘newoid1’’ has been added as ‘‘1.2.3.4.1’’.

ENVIRONMENT

OPENSSL_CONF

The path to the config file. Ignored in set-user-ID and set-group-ID programs.

OPENSSL_ENGINES

The path to the engines directory. Ignored in set-user-ID and set-group-ID programs.

BUGS

Currently there is no way to include characters using the octal **\nnn** form. Strings are all null terminated so nulls cannot form part of the value.

The escaping isn't quite right: if you want to use sequences like **\n** you can't use any quote escaping on the same line.

Files are loaded in a single pass. This means that a variable expansion will only work if the variables referenced are defined earlier in the file.

SEE ALSO

x509 (1), **req** (1), **ca** (1)

COPYRIGHT