NAME

x509v3_config − X509 V3 certificate extension configuration format

DESCRIPTION

Several of the OpenSSL utilities can add extensions to a certificate or certificate request based on the contents of a configuration file.

Typically the application will contain an option to point to an extension section. Each line of the extension section takes the form:

```
extension_name=[critical,] extension_options
```

If **critical** is present then the extension will be critical.

The format of **extension_options** depends on the value of **extension_name**.

There are four main types of extension: *string* extensions, *multi-valued* extensions, *raw* and *arbitrary* extensions.

String extensions simply have a string which contains either the value itself or how it is obtained.

For example:

```
nsComment="This is a Comment"
```

Multi-valued extensions have a short form and a long form. The short form is a list of names and values:

```
basicConstraints=critical,CA:true,pathlen:1
```

The long form allows the values to be placed in a separate section:

```
basicConstraints=critical,@bs_section

[bs_section]

CA=true
pathlen=1
```

Both forms are equivalent.

The syntax of raw extensions is governed by the extension code: it can for example contain data in multiple sections. The correct syntax to use is defined by the extension code itself: check out the certificate policies extension for an example.

If an extension type is unsupported then the *arbitrary* extension syntax must be used, see the ARBITRARY EXTENSIONS section for more details.

STANDARD EXTENSIONS

The following sections describe each supported extension in detail.

Basic Constraints.

This is a multi valued extension which indicates whether a certificate is a CA certificate. The first (mandatory) name is **CA** followed by **TRUE** or **FALSE**. If **CA** is **TRUE** then an optional **pathlen** name followed by a nonnegative value can be included.

For example:

```
basicConstraints=CA:TRUE

basicConstraints=CA:FALSE

basicConstraints=critical,CA:TRUE, pathlen:0
```

A CA certificate **must** include the basicConstraints value with the CA field set to TRUE. An end user certificate must either set CA to FALSE or exclude the extension entirely. Some software may require the inclusion of basicConstraints with CA set to FALSE for end entity certificates.

The pathlen parameter indicates the maximum number of CAs that can appear below this one in a chain. So

if you have a CA with a pathlen of zero it can only be used to sign end user certificates and not further CAs.

Key Usage.

Key usage is a multi valued extension consisting of a list of names of the permitted key usages.

The supported names are: digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly and decipherOnly.

Examples:

```
keyUsage=digitalSignature, nonRepudiation

keyUsage=critical, keyCertSign
```

Extended Key Usage.

This extensions consists of a list of usages indicating purposes for which the certificate public key can be used for,

These can either be object short names or the dotted numerical form of OIDs. While any OID can be used only certain values make sense. In particular the following PKIX, NS and MS values are meaningful:

Value	Meaning
serverAuth	SSL/TLS Web Server Authentication.
clientAuth	SSL/TLS Web Client Authentication.
codeSigning	Code signing.
emailProtection	E-mail Protection (S/MIME).
timeStamping	Trusted Timestamping
OCSPSigning	OCSP Signing
ipsecIKE	ipsec Internet Key Exchange
msCodeInd	Microsoft Individual Code Signing (authenticode)
msCodeCom	Microsoft Commercial Code Signing (authenticode)
msCTLSign	Microsoft Trust List Signing
msEFS	Microsoft Encrypted File System

Examples:

```
extendedKeyUsage=critical,codeSigning,1.2.3.4
extendedKeyUsage=serverAuth,clientAuth
```

Subject Key Identifier.

This is really a string extension and can take two possible values. Either the word **hash** which will automatically follow the guidelines in RFC3280 or a hex string giving the extension value to include. The use of the hex string is strongly discouraged.

Example:

```
subjectKeyIdentifier=hash
```

Authority Key Identifier.

The authority key identifier extension permits two options. keyid and issuer: both can take the optional value ‘‘always’’.

If the keyid option is present an attempt is made to copy the subject key identifier from the parent certificate. If the value ‘‘always’’ is present then an error is returned if the option fails.

The issuer option copies the issuer and serial number from the issuer certificate. This will only be done if the keyid option fails or is not included unless the ‘‘always’’ flag will always include the value.

Example:

```
authorityKeyIdentifier=keyid,issuer
```

Subject Alternative Name.

The subject alternative name extension allows various literal values to be included in the configuration file. These include **email** (an email address) **URI** a uniform resource indicator, **DNS** (a DNS domain name), **RID** (a registered ID: OBJECT IDENTIFIER), **IP** (an IP address), **dirName** (a distinguished name) and otherName.

The email option include a special 'copy' value. This will automatically include any email addresses contained in the certificate subject name in the extension.

The IP address used in the **IP** options can be in either IPv4 or IPv6 format.

The value of **dirName** should point to a section containing the distinguished name to use as a set of name value pairs. Multi values AVAs can be formed by prefacing the name with a **+** character.

otherName can include arbitrary data associated with an OID: the value should be the OID followed by a semicolon and the content in standard **ASN1_generate_nconf** (3) format.

Examples:

```
subjectAltName=email:copy,email:my@other.address,URI:http://my.url.here/
subjectAltName=IP:192.168.7.1
subjectAltName=IP:13::17
subjectAltName=email:my@other.address,RID:1.2.3.4
subjectAltName=otherName:1.2.3.4;UTF8:some other identifier

subjectAltName=dirName:dir_sect

[dir_sect]
C=UK
O=My Organization
OU=My Unit
CN=My Name
```

Issuer Alternative Name.

The issuer alternative name option supports all the literal options of subject alternative name. It does **not** support the email:copy option because that would not make sense. It does support an additional issuer:copy option that will copy all the subject alternative name values from the issuer certificate (if possible).

Example:

```
issuerAltName = issuer:copy
```

Authority Info Access.

The authority information access extension gives details about how to access certain information relating to the CA. Its syntax is accessOID;location where *location* has the same syntax as subject alternative name (except that email:copy is not supported). accessOID can be any valid OID but only certain values are meaningful, for example OCSP and caIssuers.

Example:

```
authorityInfoAccess = OCSP;URI:http://ocsp.my.host/
authorityInfoAccess = caIssuers;URI:http://my.ca/ca.html
```

CRL distribution points

This is a multi-valued extension whose options can be either in name:value pair using the same form as subject alternative name or a single value representing a section name containing all the distribution point fields.

For a name:value pair a new DistributionPoint with the fullName field set to the given value both the cRLissuer and reasons fields are omitted in this case.

In the single option case the section indicated contains values for each field. In this section:

If the name is ''fullname'' the value field should contain the full name of the distribution point in the same

format as subject alternative name.

If the name is ‘‘relativename’’ then the value field should contain a section name whose contents represent a DN fragment to be placed in this field.

The name ‘‘CRLIssuer’’ if present should contain a value for this field in subject alternative name format.

If the name is ‘‘reasons’’ the value field should consist of a comma separated field containing the reasons. Valid reasons are: ‘‘keyCompromise’’, ‘‘CACompromise’’, ‘‘affiliationChanged’’, ‘‘superseded’’, ‘‘cessationOfOperation’’, ‘‘certificateHold’’, ‘‘privilegeWithdrawn’’ and ‘‘AACompromise’’.

Simple examples:

```
crlDistributionPoints=URI:http://myhost.com/myca.crl
crlDistributionPoints=URI:http://my.com/my.crl,URI:http://oth.com/my.crl
```

Full distribution point example:

```
crlDistributionPoints=crldp1_section

[crldp1_section]

fullname=URI:http://myhost.com/myca.crl
CRLissuer=dirName:issuer_sect
reasons=keyCompromise, CACompromise

[issuer_sect]
C=UK
O=Organisation
CN=Some Name
```

Issuing Distribution Point

This extension should only appear in CRLs. It is a multi valued extension whose syntax is similar to the ‘‘section’’ pointed to by the CRL distribution points extension with a few differences.

The names ‘‘reasons’’ and ‘‘CRLissuer’’ are not recognized.

The name ‘‘onlysomereasons’’ is accepted which sets this field. The value is in the same format as the CRL distribution point ‘‘reasons’’ field.

The names ‘‘onlyuser’’, ‘‘onlyCA’’, ‘‘onlyAA’’ and ‘‘indirectCRL’’ are also accepted the values should be a boolean value (TRUE or FALSE) to indicate the value of the corresponding field.

Example:

```
issuingDistributionPoint=critical, @idp_section

[idp_section]

fullname=URI:http://myhost.com/myca.crl
indirectCRL=TRUE
onlysomereasons=keyCompromise, CACompromise

[issuer_sect]
C=UK
O=Organisation
CN=Some Name
```

Certificate Policies.

This is a *raw* extension. All the fields of this extension can be set by using the appropriate syntax.

If you follow the PKIX recommendations and just using one OID then you just include the value of that OID. Multiple OIDs can be set separated by commas, for example:

```
certificatePolicies= 1.2.4.5, 1.1.3.4
```

If you wish to include qualifiers then the policy OID and qualifiers need to be specified in a separate section: this is done by using the `@section` syntax instead of a literal OID value.

The section referred to must include the policy OID using the name policyIdentifier, cPSuri qualifiers can be included using the syntax:

```
CPS.nnn=value
```

userNotice qualifiers can be set using the syntax:

```
userNotice.nnn=@notice
```

The value of the userNotice qualifier is specified in the relevant section. This section can include explicitText, organization and noticeNumbers options. explicitText and organization are text strings, noticeNumbers is a comma separated list of numbers. The organization and noticeNumbers options (if included) must BOTH be present. If you use the userNotice option with IE5 then you need the 'ia5org' option at the top level to modify the encoding: otherwise it will not be interpreted properly.

Example:

```
certificatePolicies=ia5org,1.2.3.4,1.5.6.7.8,@polsect

[polsect]

policyIdentifier = 1.3.5.8
CPS.1="http://my.host.name/"
CPS.2="http://my.your.name/"
userNotice.1=@notice

[notice]

explicitText="Explicit Text Here"
organization="Organisation Name"
noticeNumbers=1,2,3,4
```

The **ia5org** option changes the type of the *organization* field. In RFC2459 it can only be of type DisplayText. In RFC3280 IA5String is also permissible. Some software (for example some versions of MSIE) may require ia5org.

ASN1 type of explicitText can be specified by prepending **UTF8**, **BMP** or **VISIBLE** prefix followed by colon. For example:

```
[notice]
explicitText="UTF8:Explicit Text Here"
```

Policy Constraints

This is a multi-valued extension which consisting of the names **requireExplicitPolicy** or **inhibitPolicyMapping** and a non negative integer value. At least one component must be present.

Example:

```
policyConstraints = requireExplicitPolicy:3
```

Inhibit Any Policy

This is a string extension whose value must be a non negative integer.

Example:

```
inhibitAnyPolicy = 2
```

Name Constraints

The name constraints extension is a multi-valued extension. The name should begin with the word **permitted** or **excluded** followed by a **;**. The rest of the name and the value follows the syntax of

subjectAltName except email:copy is not supported and the **IP** form should consist of an IP addresses and subnet mask separated by a /.

Examples:

```
nameConstraints=permitted;IP:192.168.0.0/255.255.0.0

nameConstraints=permitted;email:.somedomain.com

nameConstraints=excluded;email:.com
```

OCSP No Check

The OCSP No Check extension is a string extension but its value is ignored.

Example:

```
noCheck = ignored
```

TLS Feature (aka Must Staple)

This is a multi-valued extension consisting of a list of TLS extension identifiers. Each identifier may be a number (0..65535) or a supported name. When a TLS client sends a listed extension, the TLS server is expected to include that extension in its reply.

The supported names are: **status_request** and **status_request_v2**.

Example:

```
tlsfeature = status_request
```

DEPRECATED EXTENSIONS

The following extensions are non standard, Netscape specific and largely obsolete. Their use in new applications is discouraged.

Netscape String extensions.

Netscape Comment (**nsComment**) is a string extension containing a comment which will be displayed when the certificate is viewed in some browsers.

Example:

```
nsComment = "Some Random Comment"
```

Other supported extensions in this category are: **nsBaseUrl**, **nsRevocationUrl**, **nsCaRevocationUrl**, **nsRenewalUrl**, **nsCaPolicyUrl** and **nsSslServerName**.

Netscape Certificate Type

This is a multi-valued extensions which consists of a list of flags to be included. It was used to indicate the purposes for which a certificate could be used. The basicConstraints, keyUsage and extended key usage extensions are now used instead.

Acceptable values for nsCertType are: **client**, **server**, **email**, **objsign**, **reserved**, **sslCA**, **emailCA**, **objCA**.

ARBITRARY EXTENSIONS

If an extension is not supported by the OpenSSL code then it must be encoded using the arbitrary extension format. It is also possible to use the arbitrary format for supported extensions. Extreme care should be taken to ensure that the data is formatted correctly for the given extension type.

There are two ways to encode arbitrary extensions.

The first way is to use the word ASN1 followed by the extension content using the same syntax as **ASN1_generate_nconf** (3). For example:

```
1.2.3.4=critical,ASN1:UTF8String:Some random data

1.2.3.4=ASN1:SEQUENCE:seq_sect

[seq_sect]
```

```
field1 = UTF8:field1
field2 = UTF8:field2
```

It is also possible to use the word DER to include the raw encoded data in any extension.

```
1.2.3.4=critical,DER:01:02:03:04
1.2.3.4=DER:01020304
```

The value following DER is a hex dump of the DER encoding of the extension Any extension can be placed in this form to override the default behaviour. For example:

```
basicConstraints=critical,DER:00:01:02:03
```

WARNINGS

There is no guarantee that a specific implementation will process a given extension. It may therefore be sometimes possible to use certificates for purposes prohibited by their extensions because a specific application does not recognize or honour the values of the relevant extensions.

The DER and ASN1 options should be used with caution. It is possible to create totally invalid extensions if they are not used carefully.

NOTES

If an extension is multi-value and a field value must contain a comma the long form must be used otherwise the comma would be misinterpreted as a field separator. For example:

```
subjectAltName=URI:ldap://somehost.com/CN=foo,OU=bar
```

will produce an error but the equivalent form:

```
subjectAltName=@subject_alt_section

[subject_alt_section]
subjectAltName=URI:ldap://somehost.com/CN=foo,OU=bar
```

is valid.

Due to the behaviour of the OpenSSL **conf** library the same field name can only occur once in a section. This means that:

```
subjectAltName=@alt_section

[alt_section]

email=steve@here
email=steve@there
```

will only recognize the last value. This can be worked around by using the form:

```
[alt_section]

email.1=steve@here
email.2=steve@there
```

SEE ALSO

req (1), **ca** (1), **x509** (1), **ASN1_generate_nconf** (3)

COPYRIGHT