NAME

x509v3_config - X509 V3 certificate extension configuration format

DESCRIPTION

Several OpenSSL commands can add extensions to a certificate or certificate request based on the contents of a configuration file and CLI options such as **-addext**. The syntax of configuration files is described in **config**(5). The commands typically have an option to specify the name of the configuration file, and a section within that file; see the documentation of the individual command for details.

This page uses **extensions** as the name of the section, when needed in examples.

Each entry in the extension section takes the form:

```
 name = [critical, ]value(s)
```

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

If multiple entries are processed for the same extension name, later entries override earlier ones with the same name.

The format of **values** depends on the value of **name**, many have a type-value pairing where the type and value are separated by a colon. There are four main types of extension:

```
 string
 multi-valued
 raw
 arbitrary
```

Each is described in the following paragraphs.

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

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

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

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

```
 [extensions]
 basicConstraints = critical, @basic_constraints

 [basic_constraints]
 CA = true
 pathlen = 1
```

Both forms are equivalent.

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:

```
 [extensions]
 subjectAltName = @subject_alt_section

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

is valid.

OpenSSL does not support multiple occurrences of the same field within a section. In this example:

```
 [extensions]
 subjectAltName = @alt_section

 [alt_section]
 email = steve@example.com
 email = steve@example.org
```

will only recognize the last value. To specify multiple values append a numeric identifier, as shown here:

```
 [extensions]
 subjectAltName = @alt_section

 [alt_section]
 email.1 = steve@example.com
 email.2 = steve@example.org
```

The syntax of raw extensions is defined by the source code that parses the extension but should be documented. See "Certificate Policies" for an example of a raw extension.

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 the syntax of each supported extension. They do not define the semantics of the extension.

Basic Constraints

This is a multi-valued extension which indicates whether a certificate is a CA certificate. The first value 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:1
```

A CA certificate *must* include the **basicConstraints** name with the **CA** parameter set to **TRUE**. An end-user certificate must either have **CA:FALSE** or omit the extension entirely. The **pathlen** parameter specifies the maximum number of CAs that can appear below this one in a chain. A **pathlen** of zero means the CA cannot sign any sub-CA's, and can only sign end-entity certificates.

Key Usage

Key usage is a multi-valued extension consisting of a list of names of the permitted key usages. The defined values are: "digitalSignature", "nonRepudiation", "keyEncipherment", "dataEncipherment", "keyAgreement", "keyCertSign", "cRLSign", "encipherOnly", and "decipherOnly".

Examples:

```
 keyUsage = digitalSignature, nonRepudiation
```

```
keyUsage = critical, keyCertSign
```

Extended Key Usage

This extension consists of a list of values indicating purposes for which the certificate public key can be used. Each value can be either a short text name or an OID. The following text names, and their intended meaning, are known:

Value	Meaning according to RFC 5280 etc.
serverAuth	SSL/TLS WWW Server Authentication
clientAuth	SSL/TLS WWW 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

While IETF RFC 5280 says that **id-kp-serverAuth** and **id-kp-clientAuth** are only for WWW use, in practice they are used for all kinds of TLS clients and servers, and this is what OpenSSL assumes as well.

Examples:

```
extendedKeyUsage = critical, codeSigning, 1.2.3.4

extendedKeyUsage = serverAuth, clientAuth
```

Subject Key Identifier

The SKID extension specification has a value with three choices. If the value is the word **none** then no SKID extension will be included. If the value is the word **hash**, or by default for the **x509**, **req**, and **ca** apps, the process specified in RFC 5280 section 4.2.1.2. (1) is followed: The keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits).

Otherwise, the value must be a hex string (possibly with ":" separating bytes) to output directly, however, this is strongly discouraged.

Example:

```
 subjectKeyIdentifier = hash
```

Authority Key Identifier

The AKID extension specification may have the value **none** indicating that no AKID shall be included. Otherwise it may have the value **keyid** or **issuer** or both of them, separated by ",". Either or both can have the option **always**, indicated by putting a colon ":" between the value and this option. For self-signed certificates the AKID is suppressed unless **always** is present. By default the **x509**, **req**, and **ca** apps behave as if "none" was given for self-signed certificates and "keyid, issuer" otherwise.

If **keyid** is present, an attempt is made to copy the subject key identifier (SKID) from the issuer certificate except if the issuer certificate is the same as the current one and it is not self-signed. The hash of the public key related to the signing key is taken as fallback if the issuer certificate is the same as the current certificate. If **always** is present but no value can be obtained, an error is returned.

If **issuer** is present, and in addition it has the option **always** specified or **keyid** is not present, then the issuer DN and serial number are copied from the issuer certificate.

Examples:

```
 authorityKeyIdentifier = keyid, issuer

 authorityKeyIdentifier = keyid, issuer:always
```

Subject Alternative Name

This is a multi-valued extension that supports several types of name identifier, including **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 syntax of each is described in the following paragraphs.

The **email** option has two special values. "copy" will automatically include any email addresses contained in the certificate subject name in the extension. "move" will automatically move any email addresses from the certificate subject name to the extension.

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

The value of **dirName** is specifies the configuration section containing the distinguished name to use, as a set of name-value pairs. Multi-valued AVAs can be formed by prefacing the name with a **+** character.

The value of **otherName** can include arbitrary data associated with an OID; the value should be the OID followed by a semicolon and the content in specified using the syntax in **ASN1_generate_nconf**(3).

Examples:

```
subjectAltName = email:copy, email:my@example.com, URI:http://my.example.com/

subjectAltName = IP:192.168.7.1

subjectAltName = IP:13::17

subjectAltName = email:my@example.com, RID:1.2.3.4

subjectAltName = otherName:1.2.3.4;UTF8:some other identifier

[extensions]
subjectAltName = dirName:dir_sect

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

Non-ASCII Email Address conforming the syntax defined in Section 3.3 of RFC 6531 are provided as otherName.SmtpUTF8Mailbox. According to RFC 8398, the email address should be provided as UTF8String. To enforce the valid representation in the certificate, the SmtpUTF8Mailbox should be provided as follows

```
subjectAltName=@alts
[alts]
otherName = 1.3.6.1.5.5.7.8.9;FORMAT:UTF8,UTF8String:nonasciiname.example.com
```

Issuer Alternative Name

This extension supports most of the options of subject alternative name; it does not support **email:copy**. It also adds **issuer:copy** as an allowed value, which copies any subject alternative names from the issuer certificate, if possible.

Example:

```
 issuerAltName = issuer:copy
```

Authority Info Access

This extension gives details about how to retrieve information that related to the certificate that the CA makes available. The syntax is **access_id;location**, where **access_id** is an object identifier (although only a few values are well-known) and **location** has the same syntax as subject alternative name (except that **email:copy** is not supported).

Possible values for access_id include **OCSP** (OCSP responder), **caIssuers** (CA Issuers), **ad_timestamping** (AD Time Stamping), **AD_DVCS** (ad dvcs), **caRepository** (CA Repository).

Examples:

```
 authorityInfoAccess = OCSP;URI:http://ocsp.example.com/,caIssuers;URI:http://myca.example.com/ca.cer

 authorityInfoAccess = OCSP;URI:http://ocsp.example.com/
```

CRL distribution points

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

When a name-value pair is used, a DistributionPoint extension will be set with the given value as the fullName field as the distributionPoint value, and the reasons and cRLIssuer fields will be omitted.

When a single option is used, the value specifies the section, and that section can have the following items:

fullname

The full name of the distribution point, in the same format as the subject alternative name.

relativename

The value is taken as a distinguished name fragment that is set as the value of the nameRelativeToCRLIssuer field.

CRLIssuer

The value must in the same format as the subject alternative name.

reasons

A multi-value field that contains the reasons for revocation. The recognized values are:

"keyCompromise", "CACompromise", "affiliationChanged", "superseded", "cessationOfOperation", "certificateHold", "privilegeWithdrawn", and "AACompromise".

Only one of **fullname** or **relativename** should be specified.

Simple examples:

```
 crlDistributionPoints = URI:http://example.com/myca.crl

 crlDistributionPoints = URI:http://example.com/myca.crl, URI:http://example.org/my.crl
```

Full distribution point example:

```
 [extensions]
 crlDistributionPoints = crldp1_section

 [crldp1_section]
 fullname = URI:http://example.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. The following names have meaning:

fullname

The full name of the distribution point, in the same format as the subject alternative name.

relativename

The value is taken as a distinguished name fragment that is set as the value of the nameRelativeToCRLIssuer field.

onlysomereasons

A multi-value field that contains the reasons for revocation. The recognized values are: "keyCompromise", "CACompromise", "affiliationChanged", "superseded", "cessationOfOperation", "certificateHold", "privilegeWithdrawn", and "AACompromise".

onlyuser, onlyCA, onlyAA, indirectCRL

The value for each of these names is a boolean.

Example:

```
[extensions]
issuingDistributionPoint = critical, @idp_section

[idp_section]
fullname = URI:http://example.com/myca.crl
indirectCRL = TRUE
onlysomereasons = keyCompromise, CACompromise
```

Certificate Policies

This is a *raw* extension that supports all of the defined fields of the certificate extension.

Policies without qualifiers are specified by giving the OID. Multiple policies are comma-separated. For example:

```
certificatePolicies = 1.2.4.5, 1.1.3.4
```

To include policy qualifiers, use the "@section" syntax to point to a section that specifies all the information.

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

```
CPS.nnn = value
```

where "nnn" is a number.

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. Some software might require the **ia5org** option at the top level; this changes the encoding from Displaytext to IA5String.

Example:

```
 [extensions]
 certificatePolicies = ia5org, 1.2.3.4, 1.5.6.7.8, @polsect

 [polsect]
 policyIdentifier = 1.3.5.8
 CPS.1 = "http://my.host.example.com/"
 CPS.2 = "http://my.your.example.com/"
 userNotice.1 = @notice

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

The character encoding of explicitText can be specified by prefixing the value with **UTF8**, **BMP**, or **VISIBLE** 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

This 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:.example.com

 nameConstraints = excluded;email:.com
```

OCSP No Check

This is a string extension. It is parsed, but 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. Other extensions of this type 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:

```
 [extensions]
 1.2.3.4 = critical, ASN1:UTF8String:Some random data
 1.2.3.4.1 = 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
 1.2.3.4.1 = 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 invalid extensions if they are not used carefully.

SEE ALSO

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

COPYRIGHT