NAME

property − Properties, a selection mechanism for algorithm implementations

DESCRIPTION

As of OpenSSL 3.0, a new method has been introduced to decide which of multiple implementations of an algorithm will be used. The method is centered around the concept of properties. Each implementation defines a number of properties and when an algorithm is being selected, filters based on these properties can be used to choose the most appropriate implementation of the algorithm.

Properties are like variables, they are referenced by name and have a value assigned.

Property Names

Property names fall into two categories: those reserved by the OpenSSL project and user defined names. A *reserved* property name consists of a single C−style identifier (except for leading underscores not being permitted), which begins with a letter and can be followed by any number of letters, numbers and underscores. Property names are case-insensitive, but OpenSSL will only use lowercase letters.

A *user defined* property name is similar, but it **must** consist of two or more C−style identifiers, separated by periods. The last identifier in the name can be considered the 'true' property name, which is prefixed by some sort of 'namespace'. Providers for example could include their name in the prefix and use property names like

```
<provider_name>.<property_name>
<provider_name>.<algorithm_name>.<property_name>
```

Properties

A *property* is a *name=value* pair. A*pr operty definition* is a sequence of comma separated properties. There can be any number of properties in a definition, however each name must be unique. For example: "" defines an empty property definition (i.e., no restriction); ''my.foo=bar" defines a property named *my.foo* which has a string value *bar* and ''iteration.count=3'' defines a property named *iteration.count* which has a numeric value of *3*. The full syntax for property definitions appears below.

Implementations

Each implementation of an algorithm can define any number of properties. For example, the default provider defines the property *provider=default* for all of its algorithms. Likewise, OpenSSL's FIPS provider defines *provider=fips* and the legacy provider defines *provider=legacy* for all of their algorithms.

Queries

A *property query clause* is a single conditional test. For example, ''fips=yes'', ''provider!=default'' or ''?iteration.count=3''. The first two represent mandatory clauses, such clauses **must** match for any algorithm to even be under consideration. The third clause represents an optional clause. Matching such clauses is not a requirement, but any additional optional match counts in favor of the algorithm. More details about that in the **Lookups** section. A *property query* is a sequence of comma separated property query clauses. It is an error if a property name appears in more than one query clause. The full syntax for property queries appears below, but the available syntactic features are:

- **=** is an infix operator providing an equality test.
- **!=** is an infix operator providing an inequality test.
- **?** is a prefix operator that means that the following clause is optional but preferred.
- **−** is a prefix operator that means any global query clause involving the following property name should be ignored.
- **"..."** is a quoted string. The quotes are not included in the body of the string.
- **'...'** is a quoted string. The quotes are not included in the body of the string.

Lookups

When an algorithm is looked up, a property query is used to determine the best matching algorithm. All mandatory query clauses **must** be present and the implementation that additionally has the largest number of matching optional query clauses will be used. If there is more than one such optimal candidate, the

result will be chosen from amongst those in an indeterminate way. Ordering of optional clauses is not significant.

Shortcut

In order to permit a more concise expression of boolean properties, there is one short cut: a property name alone (e.g. ‘‘my.property’’) is exactly equivalent to ‘‘my.property=yes’’ in both definitions and queries.

Global and Local

Two levels of property query are supported. A context based property query that applies to all fetch operations and a local property query. Where both the context and local queries include a clause with the same name, the local clause overrides the context clause.

It is possible for a local property query to remove a clause in the context property query by preceding the property name with a ’−’. For example, a context property query that contains ‘‘fips=yes’’ would normally result in implementations that have ‘‘fips=yes’’.

However, if the setting of the ‘‘fips’’ property is irrelevant to the operations being performed, the local property query can include the clause ‘‘−fips’’. Note that the local property query could not use ‘‘fips=no’’ because that would disallow any implementations with ‘‘fips=yes’’ rather than not caring about the setting.

SYNTAX

The lexical syntax in EBNF is given by:

```
Definition     ::= PropertyName ( '=' Value )?
                       ( ',' PropertyName ( '=' Value )? )*
Query          ::= PropertyQuery ( ',' PropertyQuery )*
PropertyQuery  ::= '−' PropertyName
                 | '?'? ( PropertyName (( '=' | '!=' ) Value)?)
Value          ::= NumberLiteral | StringLiteral
StringLiteral  ::= QuotedString | UnquotedString
QuotedString   ::= '"' [^"]* '"' | "'" [^']* "'"
UnquotedString ::= [^{space},]+
NumberLiteral  ::= '0' ( [0−7]* | 'x' [0−9A−Fa−f]+ ) | '−'? [1−9] [0−9]+
PropertyName   ::= [A−Z] [A−Z0−9_]* ( '.' [A−Z] [A−Z0−9_]* )*
```

HISTORY

Properties were added in OpenSSL 3.0

COPYRIGHT