XXXXX(d) : 2018

CONFIGURATION MANAGEMENT — NEREON CONFIGURATION MODEL — OBJECT CONFIGURATION AND SCHEMA SYNTAX

Ribose Group Inc. 2018

RIBOSE STANDARD

DRAFT STANDARD

WARNING FOR DRAFTS

This document is not a Ribose Standard. It is distributed for review and comment, and is subject to change without notice and may not be referred to as a Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

CONTENTS

FOREWORD

Ribose is the asymmetric security company.

Ribose Group Inc. ("Ribose") is global developer of *asymmetric security* technologies across user-centric systems and applications.

Ribose works closely with international organizations such as ISO, CalConnect and the Cloud Security Alliance.

The procedures used to develop this document and those intended for its further maintenance are described in the Ribose Standardization Directives.

In particular the different approval criteria needed for the different types of Ribose documents should be noted. This document was drafted in accordance with the editorial rules of the Ribose Standardization Directives.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. Ribose shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

This document was prepared by the Ribose Committee *Configuration management*.

INTRODUCTION

The Nereon project aims to unify and glue together cloud configuration management through the Nereon data models.

The Nereon Object Configuration syntax (NOC) and Nereon Object Configuration Schema syntax (NOS) are methods used to express configuration settings in Nereon data models, and can also be used as interchangeable files.

Nereon models: https://github.com/riboseinc/nereon-models

Nereon is a play on Nereus, the shapeshifting sea god of the Greeks, the eldest son of Pontus and Gaia. "Nereon" literally means "place of Nereus", representing the shape-shifting nature of configuration.

1. SCOPE

This document defines the Nereon Object Configuration Syntax (NOC) and Nereon Object Configuration Schema Syntax (NOS).

(TODO.)

2. NORMATIVE REFERENCES

There are no normative references in this document.

3. TERMS, DEFINITIONS, SYMBOLS AND ABBREVIATED TERMS

For the purposes of this document, the following terms and definitions apply.

3.1. Terms and definitions

3.1.1
user

person that utilizes a service

3.1.2
access control

TODO.

3.1.3
author

TODO.

3.1.4
role

TODO.

3.1.5
classification

TODO.

3.1.6
classification label

TODO.

3.1.7
content addressable storage

TODO.

3.1.8
forward secrecy

method such that user of revoked access is unable to access data created after access revocation

Note 1 to entry: Refer to section 6.3

3.1.9
public key infrastructure
PKI

TODO.

3.1.10
blockcipher

encryption algorithm that encrypts a plaintext into an equivalent sized ciphertext, using an identical key for encryption and decryption

3.2. Symbols and abbreviated terms

$E(K, m)$	Encryption of the message m using a key of an asymmetric keypair K
$D(K, m)$	Decryption of the message m using a key of an asymmetric keypair K
$BCE(K, m)$	Symmetric encryption, through the blockcipher BC, of the message m using the key K
$BCD(K, m)$	Symmetric decryption, through the blockcipher BC, of the message m using the key K

4. NEREON CONFIGURATION AND SCHEMA SYNTAX

Nereon configurations (NOC) and schemas (NOS) are described using Nereon object notation (NON). NON is a structured configuration language influenced by JSON, UCL and HCL using a syntax influenced by traditional shell interfaces.

NON templates can be defined to capture and reuse values and NON interpolation functions provide a mechanism for manipulating values when configuration is parsed.

4.1. Document Structure

NON defines three types of value; table, list and string. A table is a list of named values where names are strings. A list is an ordered list of values. A string is a textual representation of data.

NON data MUST be utf8 encoded.

A NON document comprises a single NON table value.

4.1.1. Special Characters

There are a number of characters that have special meaning within a NON document. All other characters are considered part of a string value. The special characters are:

Table 1

Character	ASCII	Meaning
SPC	32	Separates keys and values in a table and surrounds infix operators
TAB	9	Separates keys and values in a table and surrounds infix operators
(	40	Marks the start of the parameter list for interpolation functions
)	41	Marks the end of the parameter list for interpolation functions
{	123	Marks the start of a table value
}	125	Marks the end of a table value
[	91	Marks the start of a list value
]	93	Marks the end of a list value
"	34	Marks the start and end of a quoted string
#	35	Marks the beginning of a comment
$	36	Marks a template application without parameters
CR	10	Separates entries within tables, lists and parameter lists
,	44	Separates entries within tables, lists and parameter lists
\	92	Indicates an escape sequence

4.1.2. String

Strings can be either bare or quoted. A bare string comprises one or more characters not listed in the 'Special Characters' section above.

The following Special characters can be included within a bare string if they are escaped with the backslash (\ ASCII 92) character.

Table 2

Sequence	**Meaning**	**ASCII**
\(	Open brace	40
\)	Close brace	41
\{	Open curly brace	123
\}	Close curly brace	125
\[	Open square brace	91
\]	Close square brace	93
\#	Hash sign	35
\$	Dollar sign	35
\,	Comma	44
`\`	Space	32

Quoted strings are enclosed in double quotation marks (" ASCII 34) and may contain any of the `Special Characters` above with the exception of \ (ASCII 92) and " (ASCII 34). \ and " must always be escaped.

The following escape sequences are recognised in both bare and quoted strings:

Table 3

Escape Sequence	ASCII	Character represented
\n	10	Newline (Line Feed)
\r	13	Carriage Return
\t	9	Horizontal Tab
\\	92	Backslash
\'	39	Single quotation mark
\"	34	Double quotation mark
\0nn	any	The byte whose numerical value is given by 0nn interpreted as an octal number
\xhh	any	The byte whose numerical value is given by hh interpreted as a hexadecimal number
\Uhhhhhhhh	none	Unicode code point where h is a hexadecimal digit
\uhhhh	none	Unicode code point below 10000 hexadecimal

Multi-line strings are permitted as long as they are quoted.

Examples:

```
"The Rachel Papers"
Success
Money
"Time's Arrow"
```

4.1.3. List

A list is an ordered set of values. Lists are enclosed in square braces ([ ] ASCII 91/93). Within the braces are zero or more values separated by one or more comma (, ASCII 44) or newline (ASCII 10) characters. Values are numbered, starting at zero. Numbers are assigned to each value in the order in which they are defined.

Example:

```
[
    Aglovale, Breunor, Claudin
    Calogrenant, Dinadan, "Elyan the White"

    Erec, Galeschin, Gornemant,
    "Hector de Maris", Lucan,
    "Meliant de Lis", Morholt
    Safir, Segwarides, Tor
]
```

4.1.4. Table

A table is an unordered set of key and value pairs. A table is enclosed in curly braces ({ } ASCII 123/125). Within the braces are zero or more key/value pairs. Pairs are separated by one or more comma (, ASCII 44) or newline (ASCII 10) characters. Keys are strings. Table entries are defined by specifying a key and a value separated by any combination of space (`` ASCII 32) and tab (\t` ASCII 9) characters. Keys are unique with a table. If a key appears more than once in a table definition the last value is the only one retained by the table.

Example:

```
contact {
    name "John Doe"
    email {
        work "john.doe@work.domain"
        home "john@home.domain"
    }
}
```

An additional syntax is defined for table entries where multiple keys precede the value. This syntax is only permitted within a table value and has the effect of recursively defining implicit nested table values with the initial keys. The final key and the value are used to create an entry within the most deeply nested table value. Using this syntax the above example can be written as:

```
contact {
    name "John Doe"
    email work "john.doe@work.domain"
    email home "john@home.domain"
}
```

or

```
contact name "John Doe"
contact email work "john.doe@work.domain"
contact email home "john@home.domain"
```

4.1.5. Comment

The hash (`#` ASCII 35) character indicates a comment which runs up to the next CR (`\n` ASCII 10).

4.2. Interpolation

Values are interpolated when a NON document is parsed.

4.2.1. Arithmetic interpolation

Simple arithmetic is supported with the binary operators `-`, `+`, `*`, `/`, `\`, `%` and `^`. Binary operators must appear between to values and must be surrounded by one or more space (ASCII 32) or tab (ASCII 9) characters. The operators correspond respectively with the functions `subtract()`, `add()`, `multiply()`, `divide()`, `intdiv()`, `modulus()` and `power()` described below.

```
port 8000 + 80 # port "8080"
port 8000+80 # port "8000+80", probably not intentional!
```

4.2.2. Template interpolation

Template values can be defined with the `let()` construct and reused later on in the document. Template values may or may not accept arguments which will affect their expansion.

Within a template, arguments can be extracted using the `arg(n)` construct where n is the zero-based argument index.

Templates are applied by using the `apply()` construct. Templates without arguments can be applied using a special `$name` construct which is syntactic sugar for `apply(name)`.

```
let(base_port, 8000)
let(port, $base_port + arg(0))

port1 apply(port(80)) # port 8080
port2 apply(port(81)) # port 8081
port3 $base_port + 82 #port 8082
```

4.2.3. Functions

NON uses the `name(args)` construct for functions where name is a bare string and args is a comma separated list of zero or more arguments each of which is a valid NON value. The correct number of arguments will vary depending on the function.

A NON parser supports at least the following basic functions:

Table 4

Function	**Result**
`add(a, b)`	a + b
`subtract(a, b)`	a - b
`multiply(a, b)`	a × b
`divide(a, b)`	a ÷ b
`intdiv(a, b)`	a ÷ b, a and b are integers, result rounded towards zero
`modulus(a, b)`	remainder of a ÷ b, a and b are integers
`power(a, b)`	a^b

A parser may also permit user defined functions to be registered prior to parsing.

BIBLIOGRAPHY

[1] Shared generation of RSA keys, Michael Malkin, Thomas D. Wu, Dan Boneh. *Experimenting with Shared Generation of RSA keys.* NDSS 1999.