P. McCombes
T. Lunn
PA Consulting

Expires: August 2021

February 24, 2021

Simple Information Serialization Language: SISL

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on August 24, 2021.

Abstract

Cross-domain systems using the Transform-Verify approach need a simple intermediate format in which to represent structured data. The Simple Information Serialization Language (SISL) has been developed as a way of encoding data to allow efficient syntax verification in hardware. The current specification documents SISL.

Table of Contents

1. Introduction 2
2. SISL Design Principles 2
    2.1. Simplicity 3
    2.2. LL(1) Parsable 3
    2.3. Unambiguous reconstruction 3
    2.4. Maintaining order 3
    2.5. Other constraints 4
        2.5.1. Character ranges 4
            2.5.1.1. Names strings 4
            2.5.1.2. Type strings 4
            2.5.1.3. Values strings 4
                2.5.1.3.1. Escaped characters in value strings 4
        2.5.2. Nesting depth 5
    2.6. Things that are not constrained 5
3. ABNF definition of SISL 5
4. Security Considerations 6

5. IANA Considerations 6
6. References 6
6.1. Normative References 6
6.2. Informative References 7
7. Acknowledgments 7
Appendix A. Railroad diagram representation of SISL 7
Appendix B. Field lengths for the Oakdoor Import Diode 8
Appendix C. Examples of SISL files 8
A.1. Example 1: Simple SISL file 8
A.2. Example 2: SISL file showing nesting 8
A.3. Example 3: SISL file showing string escapes 9
A.4. Example 4: Longer SISL example 9

1. Introduction

Cross domain solutions use Transform-Verify for structured data [NCS2018].

Syntax verification in hardware protects subsequent software semantic verification. For example, in Figure 1 the hardware syntax verification protects the subsequent semantic verification that may be implemented in software; the semantic verification input is guaranteed to be valid SISL.

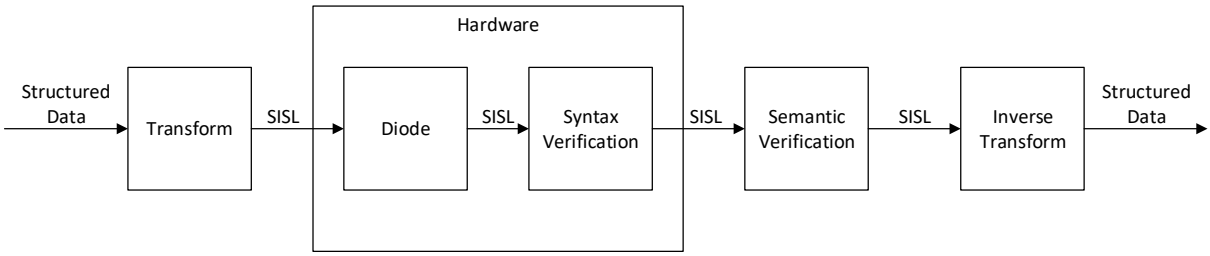


Figure 1 Cross-domain solution using Transform-Verify

Transformation is to a simple intermediate format that is relatively simple to verify in hardware, where this verification can be done fast so as not to reduce network throughput.

2. SISL Design Principles

SISL is designed to be able to represent structured data files in a way that is easy to check for syntactic validity and as a strong way to protect against binary malware appearing to be valid SISL. Examples of such structured data that can be represented in SISL include XML, JSON, CSV.

SISL is designed to be:

1. Simple
2. LL(1) parsable
3. Able to represent any structured data format in a way that can reconstruct the input unambiguously
4. Able to maintain order within groupings
5. Difficult to confuse with binary executable code

These features are described in the following subsections.

2.1. Simplicity

The first point gets lost in the detail of the ABNF specification in Section 3. There are three key rules to this very JSON-like language:

- An "element" consists of a triple made up of name, type and value with associated punctuation.

  For example:

  name: !type "value"

- A "grouping" consists of zero or more comma-separated elements surrounded by curly brackets.

  For example:

  {n1: !t1 "v1", n2: !t2 "v2"}

  where the :, ! and whitespace characters are required and the " characters must surround a value if that value is not itself a grouping. Additional whitespace is also allowed within the grouping.

  Since a grouping contains zero or more elements, a grouping can be empty. The following is a valid grouping:

  {}

- Hierarchy is enabled through nesting by allowing a value to be a grouping.

  For example:

  {n3: !t3 {n4: !t4 "v4"}}

2.2. LL(1) Parsable

LL(1) parsability allows a parser to be fast, parsing the input stream character by character with a simple and efficient implementation.

2.3. Unambiguous reconstruction

The third point on unambiguous reconstruction explains why we have a "type" field in a SISL grouping, for example {name: !type "value"}. The YAML or JSON equivalents omit the type field, so the receiver can sometimes be uncertain how to render the received value back into its original form.

2.4. Maintaining order

Arrays are represented using SISL structures, typically using hierarchical groupings. SISL maintains the ordering of elements within a grouping so that features such as YAML ordered maps can be represented.

For example, the JSON file:

```
{ "name":[ "Value 1", "Value 2", "Value 3" ] }
```

may be represented in SISL as:

```
{ name: !array {
   _: !type "Value 1",
   _: !type "Value 2",
```

_: !type "Value 3"

} }

where the ordering of the elements in the !array grouping is the same as in the JSON array.

2.5. Other constraints

Other language constraints are designed to minimize the risks to parsers:

- that binary executables can be confused as valid SISL
- that parser bugs can be exploited

2.5.1. Character ranges

The following constraints apply to the characters that make up the name, type and value strings in a SISL file.

2.5.1.1. Names strings

Name strings must start with an alpha character or one of the non-alpha - . _ characters.

Subsequent characters on the name string can be alpha, digits, or the - . _ characters.

Name strings may be constrained in length – see Appendix A.

2.5.1.2. Type strings

Type strings must start with an alpha character or one of the non-alpha - . _ characters.

Subsequent characters on the type string can be alpha, digits, or the – . _ characters.

Type strings may be constrained in length – see Appendix A.

2.5.1.3. Values strings

Values must only contain printable 7-bit ASCII characters in the range 0x20 to 0x7E, space to tilde. The characters " and \ in a value string must be escaped with a \, i.e. the following is a valid value string:

"One \"two\" three ~ four"

The following is an invalid value string:

"One "two" £ three"

since the £ sign is not a valid 7-bit ASCII character and the internal quotation marks are not escaped.

Value strings can be empty, i.e. the following is a valid value string:

""

Value strings may be constrained in length – see Appendix A.

2.5.1.3.1. Escaped characters in value strings

Only the following escaped values are allowed in value strings:

\" Quotation marks are escaped if included within value string

\\ The backslash character in value string needs to be escaped

\r Carriage return

\t Horizontal tab

\n Newline

\xHH Hexadecimal byte

\uHHHH Unicode 2-byte character

\UHHHHHHHH Unicode 4-byte character

where H is a valid hexadecimal digit [0-9A-Fa-f].

2.5.2. Nesting depth

SISL supports hierarchy through recursion, for example:

{n3: !t3 {n4: !t4, "v4"}}

where the grouping {n4: !t4, "v4"} takes the place of a value associated with n3 and t3.

The maximum nesting depth may be constrained – see Appendix A.

2.6. Things that are not constrained

SISL has no constraints such as names only appearing once in a grouping, in contrast to JSON where the names within an object should be unique. For example, the following is valid SISL:

```
{
  name: !type "Value 1",
  name: !type "Value 2",
  name: !type "Value 3"
}
```

3. ABNF definition of SISL

The following definition of SISL uses an extension of the ABNF Core Rules defined in Appendix B of RFC-5234 [RFC5234].

This ABNF definition is augmented through use of the notation *<n> to indicate that a maximum value of repetitions might be set in hardware for certain fields. Examples of such limits are discussed in Appendix A.

```
sislfile   = grouping *<n1>wsp

grouping   = "{" ( ( *<n1>wsp collection *<n1>wsp ) / *<n1>wsp ) "}"

collection = element *( "," *<n1>wsp element )

element    = name ":" 1*<n1>wsp "!" type 1*<n1>wsp value

name       = ("_" / ALPHA) *<n2>("_" / "-" / "." / ALPHA / DIGIT)

type       = ("_" / ALPHA) *<n3>("_" / "-" / "." / ALPHA / DIGIT)

value      = ( DQUOTE *<n4>(printable / escape) DQUOTE ) / grouping

escape     =  "\" ( lcr / lct / lcn / DQUOTE / "\" )

escape     =/ "\" ( (lcx 2HEXDIG) / (lcu 4HEXDIG) / (ucu 8HEXDIG) )

wsp        = SP / HTAB / CR / LF

printable  = %x20-21 / %x23-5B / %x5D-7E

          ; Printable chars apart from "" or '\'

lcr = %x72    ; lower case r

lct = %x74    ; lower case t

lcn = %x6E    ; lower case n

lcx = %x78    ; lower case x

lcu = %x75    ; lower case u

ucu = %x55    ; upper case U
```

4. Security Considerations

Malicious code can be rendered in SISL simply using techniques such as hex or base-64 encoding. An important rule for security is therefore that such encoded binary data MUST be semantically verified to verify that it is not malicious before it is reverted to binary data.

Binary executables can be written using just printable characters – the EICAR test file is an example of such an executable. The SISL structures and punctuation mitigate much of this risk but malware can be transported in value strings and must be semantically checked before being released on trusted-side networks.

5. IANA Considerations

None of which we are aware.

6. References

6.1. Normative References

[RFC5234] Crocker, D. and Overell, P. (Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC-5234, Brandenburg InternetWorking and THUS plc, January 2008.

6.2. Informative References

[NCS2018] NCSC, "Pattern: Safely Importing Data", version 1.0, 15 July 2018, https://www.ncsc.gov.uk/guidance/pattern-safely-importing-data.

[OAK2018] Oakdoor from PA Consulting Group, 2018, www.oakdoor.io

7. Acknowledgments

The authors acknowledge the absolutely critical contributions to the design of SISL made by John T, Rob D, Sean D.

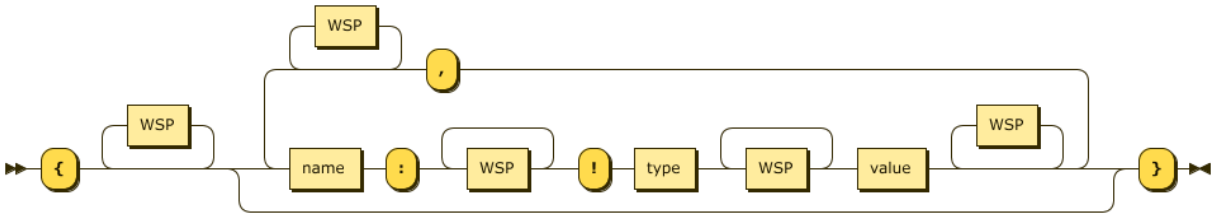
This document was prepared using 2-Word-v2.0.template.dot.

Railroad diagram representation of SISL

The ABNF definition of SISL in Section 3. can be represented by the following railroad diagrams.

A SISL file contains a SISL grouping.

A SISL grouping is:



where a value is:

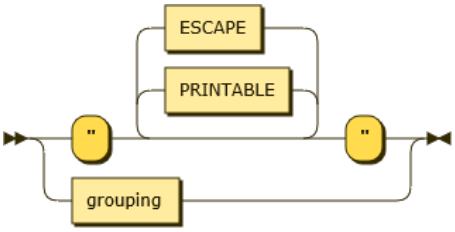


Figure 2 Railroad diagram representation of SISL

Appendix A. Field lengths for the Oakdoor Import Diode

The ABNF definition of SISL in Section 3. used the *<n> notation to indicate that certain repetition limits may exist.

This Appendix discusses the limits that are enforced in hardware by the Oakdoor Import Diode [OAK2018] hardware device.

The following values apply:

<n1>	255	White space repetition
<n2>	100 MiB	Name string length
<n3>	255	Type string length
<n4>	100 MiB	Value string length
	32	Nesting depth limit, where values are groupings
	100 MiB	File length limit

Examples of SISL files

While this RFC tries to give a precise definition of the SISL language, it is often easier to get a feel for the key rules through examples. This appendix gives some brief examples of valid SISL files.

A.1. Example 1: Simple SISL file

```
{name: !str "value"}
```

A.2. Example 2: SISL file showing nesting

```
{
  name: !str  "value",
  n2:   !obj {
    n3: !str "val1",
    n4: !str { },
    n5: !t ""
  },
  n4: !t "v",
  n5: !f {
    n6: !y "s"
  }, n: !t {n: !t {n: !t {n: !t ""
}}}}
```

A.3. Example 3: SISL file showing string escapes

```
{
  name: !str "tab \t newline \n backslash \\ quote \"",
  n: !t "carriage return \r hex \x12 unicode \u1234 Unicode \U12345678"
}
```

A.4. Example 4: Longer SISL example

```
{
  _0123456789._-: !_0123456789._- "0123456789._-",
  abcdefghijklm:  !abcdefghijklm "abcdefghijklm",
  nopqrstuvwxyz:  !nopqrstuvwxyz "nopqrstuvwxyz",
  ABCDEFGHIJKLM:  !ABCDEFGHIJKLM "ABCDEFGHIJKLM",
  NOPQRSTUVWXYZ:  !NOPQRSTUVWXYZ "NOPQRSTUVWXYZ",
  printable1: !str " !#$%&'()*+,-./:;<=",
  printable2: !str ">?@[]^_`{|}~",
  tabssp: !str   "value hello   ",
  grouping: !blah {
    name: !str "value"
  },
  nestedA: !group {
    nestedB: !group {
      nestedC: !group {
        nestedD: !group {
          nestedE: !group {
            nestedF: !group {
              something: !boring "whatever",
              and: !another "fine   then."
            }
          }
```

```
        }
      }
    }
  },
  longer_line: !s "The Quick Brown Fox Jumps Over The Lazy Dog",
  equation: !maths "a x b + c - 2 / 75 ^ 6 = nope",
  empty_group: !empty {},
  unicode_escapes: !str "\x09 \u1234 \U12345678",
  other_escapes: !str "tab \t newline \n backslash \\ quote \" carriage return \r"
}
```

Authors:

Paul McCombes
Tim Lunn

PA Consulting
Melbourn
Royston SG8 6DP
UK

Phone: +44 1763 267 323