Specification of the serialization of semantic graphs

Ferenc Domes
Kevin Kofler
Arnold Neumaier
Peter Schodl

Abstract

This document specifies a standardized textual serialization of the SM (=semantic memory) contents allowing the exchange of SM contents between different implementations of the SM.

Contents

1 Introduction 1
2 External table 2
3 Authority codes 3
4 Language codes 3
5 Dictionary 3
6 View roots 4
7 Semantic memory 4
8 Requirements 4
9 Example 4

1 Introduction

Concise can perform a standard serialization of views to .cnv files according to the following specification, and a corresponding deserialization that recovers the view from a .cnv file (even when the SM was essentially empty before loading it).

The basic structure of a serialization is divided in several parts: the external table (2), the authority codes (3), the language codes (4), the dictionary entries (5), the view roots (6) and the semantic memory (7).

There are also some additional requirements (8) and rules which each implementation of the serialization should abide.
A simple serialization example can be found in the section Example (9).
2 External table

The external table serializes each external object like names, strings, numbers, colors etc. This part of the serialization starts with the line
* EXTERNAL TABLE *
and each following line contains an entry
id=<type;value>
where the id is a unique number identifying the external value, the type is the a three character type code of the external value (nam, str, int, dbl, etc.) and the value is the string representation of the external value.

The type code of the value:
- must match the regular expression /[a-z]+/,
- must be the string ‘int’ for values of integer type,
- must be the string ‘dbl’ for values of double type,
- must be the string ‘str’ for values of string type,
- must uniquely match the type,
- in particular, may not be ‘int’, ‘dbl’ or ‘str’ for any other type.

The serialization of the value is implementation-defined, but the serialization of integers, double precision floats ans strings are defined in the following sections. Furthermore, the serialization must be valid UTF-8 and the serialization may not contain CR (\r), LF (\n) or NUL (\0) characters.

2.1 Integer Values

The serialization of an integer shall be its decimal representation without leading zeros, matching the regular expression /0|[-]?[1-9][0-9]*/. For example, 123 and -51 are valid serializations of an integer, while -0 and 0123 are not.

2.2 Double Values

The serialization of a double shall be the following format, which allows representing IEEE double-precision floating-point numbers without rounding errors. If possible, an implementation should use double-precision IEEE as its native representation for values of double types, so as to allow treatment of those values without implementation-induced rounding errors.

- NaN (Not a Number) values shall be represented as the string ‘nan’. Positive infinity shall be represented as the string ‘inf’.
- Negative infinity shall be represented as the string ‘-inf’.
- Denormal values shall be represented in a hexadecimal format matching the regular expression /-?0x0\.|[0-9A-Fa-f]{13}p-1022/, for example the string ‘0x0.23A78C8410EE8p-1022’.
- Zeros shall be represented the same way as denormals, i.e., as the string ‘0.0000000000000p-1022’ for positive zero and as the string ‘-0.0000000000000p-1022’ for negative zero. (This matches their bit representation in the IEEE floating point standard.)
- Normal values (i.e., everything else) shall be represented in a hexadecimal format matching the regular expression:
/^-?0x1\.[0-9A-Fa-f]{13}p(0|[-]?([1-9][0-9][0-9]?|10[01][0-9][0-9]?|102[0-2]))|1023)/, for example the string ‘-0x1.23A78C8410EE8p-259’.

(In the above formats, the ‘p’ character is used to denote ‘times 2 to the power’, i.e., a binary exponent, as in ISO C99 hex floats.)
2.3 String Values

The serialization of a string shall be its UTF-8 representation, escaped according to the following scheme:

- The NUL (\0) character must be escaped as a backslash followed by ‘0’ (\0).
- The LF (\n) character must be escaped as a backslash followed by ‘n’ (\n).
- The CR (\r) character must be escaped as a backslash followed by ‘r’ (\r).
- The backslash character ‘\’ must be doubled (\\).

In addition, in order to facilitate manually typing the serialized representation, the following escapes may be used to encode Unicode codepoints:

- \u escapes with a 4-digit hexadecimal representation of the codepoint, matching the regular expression /\u[0-9A-Fa-f]{4}/, for example the string ‘\u4C0A’
- \U escapes with an 8-digit hexadecimal representation of the codepoint, matching the regular expression /\U[0-9A-Fa-f]{8}/, for example the string ‘\U4C0A’.

Those escapes shall be interpreted when parsing the format, but should not be generated unless generating UTF-8 is impossible for some reason.

3 Authority codes

The authority codes are used in the dictionary for assigning each name to a certain owner. This part of the serialization starts with the line

* AUTHORITY CODES *

and each following line contains an entry

aid=eid

where the aid is a unique id identifying the authority while eid is the id of the associated external value.

4 Language codes

The language codes are used in the dictionary for assigning each name to a certain language. This part of the serialization starts with the line

* LANGUAGE CODES *

and each following line contains an entry

lid=eid

where the lid is a unique id identifying the language while eid is the id of the associated external value.

5 Dictionary

The dictionary assigns to a triple of an external name, authority and language a new id. This part of the serialization starts with the line

* DICTIONARY ENTRIES *

and each following line contains an entry

eid,iid,aid,lid

where the iid is a unique id identifying the internal entry while eid,aid and lid are the associated external, authority and language ids.
6 View roots

This part of the serialization starts with the line

* VIEW ROOTS *

and the next line contains a list of comma separated of internal or external ids specifying the
roots of the serialization.

7 Semantic memory

This part encapsulates the main part of the data and starts with the line

* SEMANTIC MEMORY *

while each following line contains an entry

handle: field₁=entry₁, ... ,fieldₙ=entryₙ

where handle and all field_k are internal and all entry_k are internal or external ids. Each
line is thus a compact representation of n sems originating from the same handle handle.

7.1 Extent of the serialization

The part of the semantic memory that is to be serialized is specified in a semantic graph
view (short: SG view). A view consists of two sets of objects (the roots and the folded
objects), a set of fields (the silent fields), a set of types (the silent types), and a mapping
that associates a set of fields to certain types (the selected fields of a type).

Every sem that is reachable from one of the roots is to be serialized, except sems where every
path from every root

• contains a folded object as handle, or
• contains a silent field as field, or
• is a sem where the handle matches a silent type, or
• follows a sem where the handle matches type #T and has a field that is a selected field
of type #T.

8 Requirements

Every implementation of the serialization should abide the following basic rules:
- Neither the external table nor the dictionary should contains the special name type used
for defining the type of a Concise object. This special name is marked only by the reserved
id 1.
- For the basic authority the name System and for the basic language the name English is
reserved. Each serialization should contain these.

9 Example

Minimalist example for serialization of a single type Trailer in the type sheet TextDocument.
In this example the comments start with a % sign.

* EXTERNAL TABLE *

-1=<nam;English>
-7=<nam;System>
-424=<nam;TextDocument>
-1778=Trailer
...
* AUTHORITY CODES *
12=-7 % System
219=-424 % TextDocument
...
* LANGUAGE CODES *
120=-1 % English
...
* DICTIONARY ENTRIES *
-7,12,12,120 % System(English,System)
-1,12,12,120 % English(English,System)
-424,219,12,120 % TextDocument(English,System)
-1778,4690,219,120 % Trailer(English,TextDocument)
...
* VIEW ROOTS *
...
* SEMANTIC MEMORY *
6155:1=4690 % the type of the node #6155 is 'Trailer'
...