Specification of the DEC format

Tom Kirchner
tom@tkirchner.com

Version 1.1
August 12, 2012

Contents

1 Introduction

1.1 About this document

This document specifies a format named *DEC* that is used to store complex structured data and is therefor the official specification that is the basis for all systems that handle the DEC format.

1.2 Intention of the format

The intention behind the DEC format was that the author desired a format that would be suitable for writing programs in a purely declarative way. As a side product of this effort a format was developed that was also suited to store arbitrary complex structured data similar to XML, but less verbose and simpler. Because of its universal and simple nature, the DEC format is applicable in many cases beside beeing a format for declarative program logic.

It was then decided that the DEC format would be defined by an official specification which is this document in order to provide the means to create compatible and relieable software for handling the format.

2 Specification

2.1 Character set

The DEC format uses the Unicode character set.

2.2 Syntax

The syntax of the DEC format in EBNF (Extended Backus-Naur Form) is as follows.

```
space (ignored)              = /\s\n\r\t+/
multiline-comment (ignored)  = "/*" .. "*/"
singleline-comment (ignored) = /\#[^\n\r]*/

file                         = { declaration }
  declaration                = [ "@", identifier ], literal

literal                      = map | string | real | number | identifier
```

```
map                         = [ symbol ], "[", { pair }, "]"
  pair                      = [ symbol, ":" ], declaration
string                      = double-quoted-string | single-quoted-string
  double-quoted-string      = '"' .. '"'
  single-quoted-string      = "'" .. "'"
real                        = /\d+\.\d+/
number                      = /\d+/
identifier                  = symbol, { ".", symbol }
  symbol                    = /[\w\d]+(\-[\w\d]+)*/
```

The EBNF notation is extended by these two notational concepts to create a simpler grammar:

`/.../` This notation stands for a regular expression. It is a more flexible way of noting a terminal symbol. The regular expression notation of the Perl programming language is used (Perl regular expressions).

`"x".."y"` This notation stands for a terminal symbol that is encapsulated by two given terminal symbols `"x"` and `"y"`. Between these two terminal symbols `"y"` can only appear if it is escaped by using the escape character, which is in this case \ (backslash).

2.3 Description

DEC content is a sequence of literals. These types of literals are defined:

Number An integer/whole number of arbitrary size.

Real A real number fo arbitrary size.

String A string of UTF-8 characters.

Identifier A sequence of symbolic names that references another literal.

Map A map of other literals.

Each contained literal has a key associated. The same key can appear 0 or more times.

Keys do not have to be explicitly set but are automatically generated when ommitted. The automatically generated keys are whole numbers starting with 0 and incremented by 1 for each key that has to be automatically named.

The order of the contained literals is relevant, though the logic that interprets the DEC document may ignore the order.

A map has an optional type identifier associated. If the type identifier is ommitted, then the empty type identifier is used.

A literal has an optional name that is noted before the literal. This name is a global identifier and can be used as a literal itself.

An identifier does not have to be defined before it can be used. The logic that interprets the DEc document(s) may choose the behaviour when encountering an identifier that has not been defined after the whole DEC document has been analysed or all DEC documents have been analysed. One behaviour for example could be to create an error message, another one could be to automatically create a literal referenced by the identifier. Yet another behaviour could be to silently ignore the incident.

2.4 Examples

This is an example of DEC formatted content that is used to represent declarative program logic:

```
@t "This is a window title"
```

```
@w 256
@h w
@r 42.22
@a application [
  windows: @stuff.x [
    @mw.bla window [
      title: t
      size: size [ width: w  height: h ]
      max-size: @max size [ width: 100  height: 100 ]
      button [ model: btn ]
    ]
    bla: window []
  ]
  morestuffs: []
  @btn model [
    value: "quit"
  ]
]
```

This is an example of DEC formatted content that is used to represent a collection of contact information:

```
address-book [
  contacts: [
    contact [
      name: "Tony"
      familyname: "Baloni"
      street: "West Harvard Road"
      number: 42
      birthday: date [
        day: 21
        month: 11
        year: 1977
      ]
    ]
    contact [
      name: "Sandy"
      familyname: "Rivers"
      street: "Mainstreet"
      number: 1
      birthday: date [
        day: 11
        month: 3
        year: 1983
      ]
    ]
  ]
]
```

3 Technologies

3.1 Parsing and creating documents

Using the grammar for the DEC format, parsers can be created that read DEC formatted content into a data structure for various programming languages to use. Writers can also be created that turn data structures into DEC formatted content that conforms to the DEC grammar.

Existing DEC parser and writer implementations include (this list may be incomplete):

Data::DEC A Perl module for reading and writing DEC formatted content. This implementation is currently under development.

3.2 Structure validation of documents

Since the parser usually only validates the syntax of a DEC document, there is often a need to determine weither a given DEC document conforms to a certain structure. This leads to the DECS format, that can be used to define the structure of a class of DEC documents. See the DECS specification on detailed information.

4 References

4.1 Links

Unicode character set http://en.wikipedia.org/wiki/Unicode

XML http://de.wikipedia.org/wiki/Extensible_Markup_Language

5 Revision History

1.1 1 January 2013

Changed character set.

1.0 8 August 2012

Initial version