robbiblubber.org

Diabolic Data Notation Specification

Version 3.0

Introduction

The **Diabolic Data Notation** (ddn) is a data interchange format introduced by **robbiblubber.org** in February 2000 for the **Diabolic Device Protocol** (ddp).

It aims to be lightweight and easy to read and write for humans, as well as to be easily processable by machines. ddn is basically a hierarchical key-value-pair format. Although ddn was primarily intended to serve as a data interchange notation for ddp, it was also widely used for storing configuration data from the very beginning.

This document describes the specification version 3.0 as of April 2022. Although ddn was part of the ddp specification until version 2.0, there have been only minor changes from version 1.1. The specification changes are tracked in the version history at the end of this document.

The robbiblubber.org Spellbook Libraries offer support for ddn in a number of common programming languages.

ddn Element

In the **Diabolic Data Notation** (ddn) the basic entity is a **ddn Element**. Each element can be a **ddn Value** or a **ddn Section**. Every Element except for the root element has a name.

ddn Value

A **ddn Value** is a key-value-pair consisting of the name and an associated value. The value is a string representation of the Element's data contents that can be interpreted as various data types as needed. The Value is assigned to the Name with an equals sign, each ddn Value is terminated by a semicolon. Special characters need to be masked, both in names and values. Leading and tailing whitespaces are ignored in names and values. If a name or value starts or ends with a whitespace, these characters have to be masked.

```
name = value;
```

Definition: ddn Value

ddn Section

A **ddn Section** is a ddn Element that contains other elements. A ddn Sections starts with its name, followed by any number of ddn Elements enclosed in curly brackets. Whitespaces are ignored.

A section cannot hold a value. The root section consists only of the section contents.

```
name
{
    ...
}
```

Definition: ddn Section

ddn Root Section

The topmost element is the **ddn Root Section**. A root section does not have a name and is not enclosed in curly brackets. It only contains its child elements.

Comments

ddn supports C++ style comments in all elements.

Single-Line Comments

Single-line comments start with // (two slashes) characters and end at line break.

```
... // single line comment
```

Definition: Single-line comment

Multi-Line Comments

A multi-line comment is defined by the /* (slash, asterisk) characters, followed by any sequence of characters (including new lines), followed by the */ characters.

```
... /*
    multi-line comment */ ...
```

Definition: Multi-line comment

Masking

Special characters must be masked in ddn. Spaces need to be masked at the beginning or end of a name or value, tabs should always be masked.

There is also a mask for NULL. This is used as a value and not combined with other characters.

The following list shows the masks defined in ddn:

Character Name	Character	Mask	Remarks
Space		_	only required at beginning/end of expression
Equals sign	=	\=	unmasked as structural element
Bracket open	{	\{	unmasked as structural element
Bracket closed	}	\}	unmasked as structural element
Comma	,	\,	unmasked as structural element
Semicolon	;	\;	unmasked as structural element
Backslash	\	\\	
Slash	/	\/	
Line break		\n	
Tab		\t	
NULL		\0	represents a NULL value

Data Representation

In ddn all data is represented in string format. Consuming applications may interpret these string values as needed. The specification defines default representations that are also supported by the library implementations and will be described in the following.

Strings

String values are represented a simple strings. Quotes are not required.

```
str = This is a string;
```

Sample: a string value

Integers

Integers are represented as decimal integer strings.

```
d = 28;
```

Sample: an integer value

```
d = -1;
```

Sample: a negative integer value

Floating Point Numbers

Floating point values are represented as decimal strings.

```
pi = 3.1415926536;
```

Sample: a floating point value

Boolean Values

A Boolean is usually represent by the string expressions "true" and "false". It is also possible to use numeric values like "0" and "1".

```
ok = true;
```

Sample: a Boolean value

Date/Time Values

Date and time values are represented as timestamps with time zone in the form "yyyy-mm-dd hh:mm:ss.xxx+HH:MM" or "yyyy-mm-dd hh:mm:ss.xxx-HH:MM".

All parts may be omitted, except the date.

```
dat = 1977-08-14 15:27:22.926+02:00;
```

Sample: a date/time value

Array Values

ddn supports array values. These values consist of a comma-separated list of string-represented values. The values inside the array follow the rules described above for data representation.

```
arr1 = Peter, Paul, Mary;
```

Sample: a string array

```
arr2 = 1, 2, 3;
```

Sample: an integer array

```
arr3 = Peter, 2, Mary;
```

Sample: a mixed array

Complex Data Types

The robbiblubber.org Spellbook libraries support serialization of objects to ddn. Objects are serialized as ddn Sections containing elements for all serialized members.

In the serialization of arrays of complex types, each array element is represented by a ddn Element (typically a Section) identified by its index. Dictionaries are serialized the same way while the key is used as the Element name.

Example

```
type = T;                                        // value
time = 2022-11-19 05:33:20+00:00
winpath = \\\\svr1\\folder;
sec1                                             // section
{
   type = S;
   fab = Cotton;
   n = 20;
   sub1                                          // subsection
   {
      vals = 1.2, 1.4, 1.3;
   }
   sub2
   {
      vals = 2.3, 2.6, 1.7;
   }
}
sec2
{
   /* a multi line comment in a single line */
   type = X;
   unused = true;
}
```

Sample: ddn sample

ddn Addressing

ddn Elements are referred to by their (absolute or relative) path using the containing sections separated by a / (slash). An absolute path starts at the Eoot Element (/). . denotes the current Element, .. the parent element.

```
/sec1/sub2/vals
```

Sample: ddn path sample

Version History

Version 1.0	2000-02-16	released specification
Version 1.1	2000-03-04	support for ddn Sections
Version 1.2	2001-04-23	removed equals sign and semicolon in ddn Section declaration
Version 2.0	2009-09-01	support for array values
Version 2.1	2010-02-16	introduced masked whitespace characters
Version 3.0	2022-04-21	introduced masked character for NULL