CLOP: A Class-Based System for Managing Object Properties

Rev. 0.2 – 2009-03-23

1 Introduction

CLOP is a class-based system for managing digital object properties. "Object" is used in the most general sense to refer to any digital entity of interest. In many contexts these objects may share common properties. In cases where the number of objects is large, it is inefficient to store redundant copies of the same property values, and it can be cumbersome to affect large-scale changes to these values. The design of CLOP addresses these problems through the principle of indirection, by enforcing a clear separation of assertions by a digital object of its class *memberships* from the *definitions* of those classes. Storage space is conserved since individual objects need only keep track of a number of short class names, while the actual property values are stored once in the class definition. Global changes can be performed easily by updating the single centrally stored class definition; the change is then immediately effective for all objects claiming class membership.

While CLOP provides mechanisms for defining class properties, it does not need to understand anything about the *semantics* of those properties. Semantic interpretation of CLOP-defined class properties is the purview of processes and specifications external to CLOP.

2 Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

All uses of the term "directory" in this document MAY be interchanged with "folder" without loss of intended meaning. Similarly, all uses of the solidus ("/") as a directory path separator MAY be interchanged with a reverse solidus ("\"). In other words, this specification SHOULD be applicable in both Unix/Linux and Windows/DOS contexts.

In examples of file system directory hierarchies, non-required directories or files are enclosed by square brackets ("[" and "]"), a number sign ("#") introduces an informative comment, and an ellipsis (". . .") indicates arbitrary repetition of the previous element.

Augmented Backus-Naur Form (ABNF) [RFC5234] is used to define the syntax of specific files required or recommended by this specification. Syntax rules names left undefined in this specification (for example, `ALPHA`) SHALL be interpreted as core ABNF names.

3 Class Membership

Assertions of class membership MAY be made through two mechanisms, one appropriate for aggregate entities represented by a directory and the other appropriate for unary entities represented by individual files.

The nature of file systems mean that a unary file object must also live in some directory, so it is important to distinguish between a true aggregate object represented by the *totality* of files found in a directory hierarchy and individual unary files that are placed together in a directory for convenience.

3.1 Aggregate Class Membership

An aggregate entity is typically represented by a file system directory or directory hierarchy. Membership assertions for an aggregate entity are defined by one or more membership files found in the structural root directory for the entity. In other words, the membership assertions found in the membership file apply to the entity represented by the innermost enclosing directory.

```
<facet>-class.txt
```

Class memberships can be organized on the basis of class facets. A separate membership file is defined for each class facet.

Within a membership file individual class memberships are asserted by the class names, one per line.

```
<name1>
<name2>
...
```

3.2 Unary Class Membership

A unary entity is typically represented by an individual file, so the class memberships for unary entities are collected together into a single membership file named "`classes.txt`".

```
classes.txt
```

Within the unary membership file the various class memberships are defined on a single line associated with a specific file.

```
<file> <facet> <name1>,<name2>,...
...
```

While is it possible to use a unary membership file to assert class memberships for the individual files that make up an aggregate object, such use of CLOP is NOT RECOMMENDED. The use of the unary membership file SHOULD be reserved for unary objects.

4 Class Definition

Class definitions are found in directories named for the class facet, "`<facet>-class/`". Within this directory, a class of a given "`<name>`" is defined by a file named "`<name>.txt`".

```
<facet>-class/
              <name>.txt
```

The properties associated with the class are defined by ANVL name/value pairs [ANVL].

```
<name>: <value>
...
```

5 Implementation

TBD…

6 Security Considerations

CLOP poses no direct risk to computers or networks. As a file system convention, CLOP is capable of holding files that might contain malicious executable content, but it is no more vulnerable in this regard than any file system.

Appendix A: Complete CLOP Conventions

An aggregate object asserts its class memberships in one or more faceted membership files.

```
<facet>-class.txt
```

The production rule for facet names is:

```
<facet>      = 1*VCHAR
```

It is RECOMMENDED that short one or two character strings are used as facet names.

The production rule for aggregate membership files is:

```
<membership> = 1*(<name> EOL)
<name>       = 1*VCHAR
EOL          = CR / CRLF / LF
```

It is RECOMMENDED that class names be no longer than 32 characters.

A unary object asserts its class memberships in a membership file.

```
classes.txt
```

The production rule for unary membership files is:

```
<classes>    = 1*(<file> 1*WSP <facet> 1*WSP <name> 0*(“,” 1*WSP <name>)
                    EOL)
```

Class definitions are defined by files in a class facet directory.

```
<facet>/
        <name>
```

The production rule for class definition is:

```
<definition> = 1*(<property> “:” 1*WSP <value> EOL)
<property>   = 1*VCHAR
<value>      = 1*VHCAR
```

It is RECOMMENDED that only a single white space character (WSP) is used to separate the name/value pair.

References

[ANVL] J. Kunze, B. Kahle, J. Masanes, and G. Mohr, *A Name-Value Language (ANVL)*, Internet draft, February 14, 2005 <http://www.ietf.org/internet-draft/draft-kunze-anvl-01.txt>.

[RFC2119] Bradner, S., *Key Words for Use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, March 1997 <http://www.ietf.org/rfc/rfc2119.txt>.

[RFC5234 D. Crocker (ed.) and P. Overell, *Augmented BNF for Syntax Specifications: ABNF*, STD 68, RFC 5234, January 2008 <http://www.ietf.org/rfc/rfc5234.txt>.