eDonkey2000 protocol tag system overview

Abstract

This document describes the Tag system used in eDonkey2000 protocol.

See also:
eDonkey2000 Protocol

Table of Contents

1. Overview
2. Header specification
3. Description

1. Overview

Tags are used in ed2k network to add any number of additional data fields to existing data structures while keeping the protocol backwards compatible. The basic concept is that a tag can be identified by its data type (and/or length), and can be simply ignored if the specific tag is not supported for whatever reason.
Client's can detect and handle various tags by their opcode, which is 8-byte value. Additionally, there have been some protocol extensions which also allow 'named' tags, which - logically - have a name instead of simple opcode.

2. Header specification

Standard ed2k tag:

```
+--------+--------+--------+--------+--------
|  type  |  length = 0x01  | opcode |   type-specific amount of data
+--------+--------+--------+--------+--------
```

Extended protocol, string tag:

```
+-------+-------+-------+~~~~~~~~~~~~~~~+------
|  type |      length    | <length>name  | type-specific amount of data
+-------+-------+-------+~~~~~~~~~~~~~~~+------
```

Lugdunum extended protocol, special types

```
+-------+--------+-------
|  type | opcode | (type & 0x7f) - 0x10 bytes of string data
+-------+--------+-------
```

3. Description

As can be seen, lugdunum's extended protocol allows incorporating the string field length into the type bits, lowering the header overhead. This format can be detected by looking at the highest-order bit of type field. If it is set, the type is encoded using this extension. To retrieve the actual data type, apply &0x7f to the type. If this resolves into an integer of size 0x10 or more, the string data length can be retrieved by substracting 0x10 from the type. Otherwise, the tag should be treated as normal ed2k tag type.
Type field in the tag may be one of the following:

```
TT_HASH    = 0x01,     //!< unsupported, 16 bytes
TT_STRING  = 0x02,     //!< [u16]len[len]data
TT_UINT32  = 0x03,     //!< 4 bytes
TT_FLOAT   = 0x04,     //!< 4 bytes
TT_BOOL    = 0x05,     //!< unsupported, 1 byte
TT_BOOLARR = 0x06,     //!< unsupported, [u16]len[len]data
TT_BLOB    = 0x07,     //!< unsupported, [u16]len[len]data
TT_UINT16  = 0x08,     //!< 2 bytes
TT_UINT8   = 0x09,     //!< 1 byte
TT_BSOB    = 0x0a      //!< unsupported, [u16]len[len]data
```