Quadrilateral and Triangle Transfer Protocol – QATTP /1.0

# Status of Memo

This document specifies an Internet application protocol for storing data about polygon shape’s. The current version is a working draft subject to protocol changes; distribution of this memo is unlimited.

Contents

[*Quadrilateral and Triangle Transfer Protocol – QATTP /1.0* 1](#_Toc409898257)

[Status of Memo 1](#_Toc409898258)

[Introduction 2](#_Toc409898259)

[Purpose 2](#_Toc409898260)

[Terminology 2](#_Toc409898261)

[Protocol Constraints and Parameters 3](#_Toc409898262)

[Sockets 3](#_Toc409898263)

[Encoding 4](#_Toc409898264)

[Notation 4](#_Toc409898265)

[Protocol and Message Format 5](#_Toc409898266)

# Introduction

## Purpose

The **Quadrilateral and Triangle Transfer Protocol (QATTP)** is an application level protocol that describes an easy to use standard for storing shape data with nothing but vertex information for querying later based on a simple request and response system. It is designed to be simple, extensible, and stateless.

Distributed systems that require querying of large data-sets coming from multiple sources can benefit greatly from this protocol. The protocol allows for multiple clients to feed in data while others can query in real time. This document will outline a specification that can be used for creating compatible clients and server applications.

**QATTP** adopts a lot of policies from HTTP where it is useful and simplifies things to make the protocol lighter and minimal for the needs of the application.

## Terminology

*Shape*

A shape is described as either a ***Triangle***or ***Quadrilateral*** by the definition of this specification.

*Triangle*

Any polygon with three points is considered to be a triangle. The specification does not enforce constraints.

*Quadrilateral*

Any polygon with four points is considered to be a quadrilateral. The specification does not enforce constraints.

*Client*

For this specification, a client is a single user that is wanting to submit or look up data about shapes.

*Occurrence*

A simple count of how often a particular shape might have been counted. This is defined more in detail later.

*Query*

A query is an initial statement issued to a server.

*Filter*

A filter is a set of rules used to aid in the querying of GET shape data. See later sections for more specifics.

*Header*

A header is defined as an auxiliary piece of data attached to a request after the main query request has been constructed. These are borrowed from the HTTP specification but are *TAB* delimited rather than newline delimited like the HTTP specification.

*CRLF*

*This* is a line ending format that has been adopted for usage in this specification. It is denoted by “\r\n”

*Verb*

Similar to HTTP verbs, the only valid options are GET or POST. They allow retrieval and uploading of data, respectively.

*Shape Qualifier*

# Protocol Constraints and Parameters

## Sockets

As with most applications, **QATTP** is a server and client model application.

**QATTP** is designed as a request and response system running on a TCP socket system, with a persistent connection. Generally, a client will open one connection for all requests and all responses. Both will be serviced over the same connection, without any additional services required by the platform. The specification tries not to specify anything that requires TCP exclusively – but reliability is key when sending data to be stored, so if TCP is not used some other verification method that is not described in this specification will be required.

When a client is done communicating with the server, the client will close the connection by itself. The specification does not specify when communications are required to be terminated. However, as described shortly the server may enforce connection constraints.

Generally speaking, there is no handshake procedure for the protocol as it is an *unauthenticated* protocol. However, at the discretion of the implementation a client may be rejected connection. The specification does not specify this behavior. Instead, a client must acknowledge that the connection may be terminated at any time at the discretion of the implementation.

*Examples:* Connection lost, too many connections, not enough resources, power loss

## Encoding

All messages are transported in plain ASCII encoding text, without exception. All transmitted bytes can be treated as text according to the specification, similar to the HTTP specification.

## Notation

A brief word on notation is required before specifying the format of messages. The format is relatively simple and thus will be outlined briefly.

<REQUIRED>

A portion of a message that is shown in angled brackets such as these indicate a mandatory field. A mandatory field is always before any optional fields, outlined below.

[OPTIONAL]

A portion of a message that is shown in right brackets such as these indicate an optional field in a message. An optional field is always followed by optional fields, and nothing else.

CRLF

As described in terminology, this is a new line. This marks the end of a message.

\T HEADER:

The “\T” can be read as the standard escape sequence for the ASCII character “tab”. A portion of a message that is prefixed by the ASCII tab and suffixed with the ASCII “:”is known as a header. A header should be caps insensitive – an implementation server should normalize the input

## Protocol and Message Format

The basic protocol is described below, with emphasis on the main skeleton implementation. With no exceptions, all messages can be considered to be terminated when a blank (read: new line; CRLF) is encountered.

In general, a request from a client begins with what we call the *query statement.* The query statement follows a form similar to HTTP, in a form of:

<VERB> [SHAPE QUALIFIER/VERTICES] [\t Header1]...[\t HeaderN] CRLF

A **verb** is either **GET** or **POST** which perform operations similar to their HTTP counterparts. These are described more in detail in the client section.

In general, a response from an implementation server responds with something similar to:

<CODE> [MESSAGE] [\t Data:] CRLF

### Client

When a client wants to perform an action on the server, it must choose between two verbs. These are **GET** and **POST.** Depending on the verb, the format of the message will look different.

**GET**

When a client sends a GET request, it is making a request to fetch shape data from the server. The exact request will look like this:  
  
<VERB> [SHAPE QUALIFIER]

[\t Occurrences:]

[\t Type:]

[\t Shares:]

CRLF