



## Contents

- [1 Purpose](#)
- [2 Architectural relationships](#)
- [3 Description](#)
  - ◆ [3.1 Sessions](#)
  - ◆ [3.2 Transactions](#)
  - ◆ [3.3 Headers](#)
  - ◆ [3.4 Data suppliers](#)
  - ◆ [3.5 Filters](#)
- [4 External Links](#)
- [5 Internal Links](#)

## Purpose

The *HTTP Client API* provides a client interface for Internet applications to use the HTTP Protocol for communication with HTTP servers on the Internet. Using the API correctly enables the application to be a conditionally HTTP 1.1 compliant client, as defined in [RFC 2616](#)

Please be noted that this HTTP Client API support starts from Symbian OS 7.0s and forward. In another word, if you develop for devices such as Nokia N-Gage QD, you will do HTTP client business yourself from socket scratch.

## Architectural relationships

The HTTP Client architecture provides a generalised mechanism for HTTP-like protocols that operate over various transports. Using a single API, a client can choose an HTTP protocol, encoding and transport. The client is not expected to implement anything else specific to those choices in its own code. The default operation provides plain-text HTTP (as defined in [RFC 2616](#)) operating over a TCP/IP connection. This transport pipelines requests by default.

## Description

There are five key concepts used in the API: *sessions*, *transactions*, *headers*, *data suppliers* and *filters*.

## Sessions

A session encapsulates the client's HTTP activity over the duration of the client's execution. Usually, one

session is used at a time; however the client may use several concurrently if desired. Each session has an associated set of properties, which define the HTTP protocol, encoding and transport that are used. Those properties apply to all HTTP transactions carried out within the life of that session. The session also has an associated set of filters, that provide additional automatic behaviours on the client's behalf.

The session class is provided by `RHTTPSession`.

## Transactions

A transaction represents an interaction between the HTTP client and an HTTP origin server. Normally a transaction consists of a single exchange of messages between client and server: a client request and a server response. However, the transaction may be extended by filters that operate on it.

Transactions execute asynchronously within the client's process. The client is notified when events are available for each outstanding transaction.

Both the request and response portion of a transaction are composed of a header and an optional body. The request portion of the transaction also specifies an HTTP Method that describes the type of operation that the client wishes to invoke at the origin server, together with a URI that specifies the resource held at the server on which the method is to be invoked. The response portion of a transaction contains an HTTP status code and message, which indicate the success of the method or the state of the resource following the method. The use of request and response bodies is determined by the HTTP method in use; e.g. in error conditions, some servers may just return a status code and message, providing no entity body.

The transaction class is provided by `RHTTPTransaction`.

## Headers

The header portion of requests and responses may have zero or more fields, which are used to convey information between the HTTP client and server. The information might relate to the data conveyed in body of the message, to the actual connection between the client and server, or might be used to convey data describing the client or server themselves. Headers as transmitted to or received from the HTTP server will be in an encoded form, according to the HTTP protocol and encoding that are used. Since the HTTP API gives clients implementation independence from these choices, a generic form is used to represent header data in the API.

The headers class is provided by `RHTTPHeaders`.

## Data suppliers

The body portion of requests and responses is represented in the API as a mix-in interface, allowing the real implementation of the classes that generate body data to be wholly hidden. The API enables signalling between the client and the transport in use, to ensure that body data is only consumed or emitted at a rate the client can support. This means that clients can assemble the body of their requests piece-by-piece, have each piece transmitted only when it is ready, and be signalled when transmission is complete so the next piece may be prepared and the old one released.

Data supplier classes must implement `MHTTPDataSupplier`.

### Filters

Filters are add-on modules that provide additional behaviours to a session beyond the simple request-response transaction described above. Behaviours may be triggered by the presence of particular headers in a request or a response, by status codes in a response, or by particular events that occur on a transaction. The [RFC 2616](#) describes a number of standard behaviours that take place over a series of request-response exchanges: client authentication, redirection and caching. Client authentication and redirection are implemented as individual filters in the 7.0 release.

Since the representation of headers is generic, as described in Headers, filters may be implemented in a protocol- and transport-independent manner.

Filter classes must implement `MHTTPFilter`.

### External Links

[Hypertext Transfer Protocol -- HTTP/1.1](#)

### Internal Links

[How to send POST data to a web server](#)

[EPOCHTTP](#)

[Writing an HTTP filter plugin](#)

[Symbian C++ : Multipart/form-data](#)

--