Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 3, 2017

J. Kunze
California Digital Library
N. Nassar
Index Data ApS
May 2, 2017

THUMP: The HTTP URL Mapping Protocol
draft-kunze-thump-03

Abstract

The HTTP URL Mapping Protocol (THUMP) is a set of URL-based conventions for retrieving information and conducting searches. THUMP can be used for focused retrievals or for broad database queries. A THUMP request is a URL containing a query string that starts with a `?', and can contain one or more THUMP commands. Returned records are formatted with Dublin Core Kernel metadata as Electronic Resource Citations, which are similar to blocks of email headers.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 3, 2017.

Table of Contents

1. Overview 2
2. A Sample THUMP Session 3
3. Keys and Citations 4
4. Key-Request Dualism 5
5. Request Summary 6
5.1. Key ? help 6
5.2. Key ? was(DESCRIPTION) when(DATE) resync 6
5.3. Key ? in(DB) find(QUERY) sort([!]ELEMS) list(RANGE) show(ELEMS) as(FORMAT) 7
5.4. Key ? 9
5.5. Key ?? 9
5.6. Key ? get() put() group() apply() 9
6. Response Summary 9
7. Returned Records 10
7.1. Empty values for required elements 11
8. FAQ -- Frequently Asked Questions 12
8.1. What's the difference between THUMP, OpenSearch, SRU/SRW, and OpenURL? 12
9. Security Considerations 13
10. References 13
Authors' Addresses 14

1. Overview

This document specifies The HTTP URL Mapping Protocol (THUMP), a set of URL-based conventions for retrieving information and conducting searches. THUMP can be used for focused retrievals; e.g., for a given known-item, asking that a specifically formatted subset of information about it be returned. It can also be used for broad database queries, such as finding all records matching the word, "monitor".

A THUMP request is a URL containing a query string that starts with a `?', and can contain one or more THUMP commands. A request is passed to a server with HTTP GET (or POST if desired). The shortest request is a URL ending in `?', as in,

```
    http://example.com/object321?
```

which asks the server to return a metadata record describing the information item identified by the URL. This is a shorthand for the common request for a short description of a known-item; the completely spelled out equivalent in this case would be

```
    http://example.com/object321?show(brief)as(anvl/erc)
```

An example of a broad database search is,

```
    http://example.com/?in(books)find(war and peace)show(full)
```

Query strings and responses are UTF8-encoded [RFC3629]. A THUMP response is an HTTP message body containing one or more records. Records contain Kernel metadata [Kernel] elements h1-h4. The currently defined tag names are summarized below, formatted as Electronic Resource Citations (ERC), which are similar to blocks of email headers. In an ERC each element consists of a label, colon, and value; long values are continued on indented lines and empty lines separate records. It will be possible in a future version of THUMP to request ERC records formatted in XML.

2. A Sample THUMP Session

THUMP is very simple and follows the classical stateless HTTP communication model. This section contains a complete annotated example of a request and response exchange. To summarize, the requester sets up a TCP/HTTP session with the server system, sends a THUMP request inside an HTTP request, receives an answer inside an HTTP response, and closes the session.

In the following example THUMP session, each line has been annotated to include a line number and whether it was the client or server that sent it. Without going into depth, the session has four pieces separated by blank lines: the client's piece (lines 1-3), the server's HTTP/THUMP response headers (4-7), and the body of the server's response (8-18). The first and last lines (1 and 18) correspond to the client's steps to start the TCP session and the server's steps to end it, respectively. The heart of the request is

the known-item metadata request indicated by the URL ending in a single `?' on line 2.


```
 1  C: [opens session]
    C: GET http://ark.cdlib.org/ark:/13030/ft167nb0vq? HTTP/1.1
    C:
    S: HTTP/1.1 200 OK
 5  S: Content-Type: text/plain
    S: THUMP-Status: 0.6 200 OK
    S:
    S: erc:
    S: who:   Stanton A. Glantz and Edith D.  Balbach
10  S: what:  Tobacco War: Inside the California Battles
    S: when:  20000510
    S: where: http://ark.cdlib.org/ark:/13030/ft167nb0vq
    S: [closes session]
```

The first two server response lines (4-5) above are typical with HTTP. The next line (6) is peculiar to THUMP, and indicates the THUMP version and a normal return status. The balance of the response consists of a single metadata record (8-12) that comprises the service response.

The returned record (8-12) in this case is in the ERC format (other formats are possible). It contains four elements that answer high priority questions regarding an expression of the object: who played a major role in expressing it, what the expression was called, when is was created, and where the expression may be found.

3. Keys and Citations

A THUMP request is a command sequence operating on a Key, which is a

base URL for a service point that supports THUMP. It is expected, however, that the Key may generalize to service points in client-server computation contexts other than today's WWW.

The Key uses a "citation-centered" system of reference. This means that data elements are addressed relative to an abstract object surrogate, or "citation".

While some systems have stored metadata-based surrogates (e.g., library catalog records for books), many other systems do not. This is not an obstacle to using THUMP. The latter usually support the display or delivery of dynamically generated object citations, each consisting of such things as an access URL, a size, a date, a title, a snippet of relevant text (e.g., matching a query), plus links to related materials.

Non-surrogate information objects in this model are, loosely speaking, the priority objects for end users, and include documents, articles, books, films, recordings, etc. Surrogates, whether static

or dynamically generated, are important temporary stand-ins during discovery, filtering, and selection processes. They are easy to manipulate in large numbers because they are much more homogeneous than the objects they represent. Those objects are often too large, unwieldy, or rights-encumbered to be dealt with directly during discovery. Surrogates are also valuable in preservation since they can provide useful information about the original context, dependencies, and provenance of an object.

4. Key-Request Dualism

Although THUMP does not specify anything about the structure of the Key, it is possible for a given Key string to express, often in an ad hoc manner, information similar to that expressed in the Request query string. The more intuitive the Key structure, the greater the chance for it to carry information that might appear to repeat or even contradict commands in the Request. For example, one server might require

```
http://example.org/?in(books)find(war and peace)show(full)
```

while another server required

http://example.com/in=books/find=war+and+peace?show(full)

and a third server required

http://example.net/books/full/war_and_peace

There is a natural dualism that servers may exploit by permitting or proposing (e.g., by returning) such semantically-laden Keys. Any conventions for re-expressing THUMP commands within the Key or for resolving apparent contradictions, however, are up to individual servers and are out of scope for this document.

This document recognizes the dualism but does not constrain it except to say that for a given Key, a server that declares THUMP support MUST respond to the "help" command by listing all the commands (methods) valid for that Key. As a foundation requirement, the "help" command is a common way to ping a THUMP server to see if it is alive. As an edge case, a THUMP response might be returned even for a URL that has no request at all (not even a `?'); this might make sense, for example, when the URL serves as the base Key for an entire service.

There are cases when a server may wish to generate a temporary Key as a stand-in for a long or complex request and return it along with a subset of found records. For example, the request,

http://example.com/?in(books)find(war and peace)list(10|1)

might return the first 10 records along with a Key that could be used in subsequent requests to return the next 10 records:

http://example.com/req98765?list(10|11)

Note that this document makes no assumption about the dynamicity of queries, whether expressed partially or entirely in the Key or in the request. In either form, returned records might come from cached results or from results freshly computed upon each access. THUMP support does not constrain servers in this regard.

5. Request Summary

There are several request forms described below, with output formats listed in a later section. Spaces have been inserted for readability in the forms below; usually, inter-command spaces would not be present. It is normal to formulate THUMP queries using only a subset of the commands specified. With a few important exceptions, this document is silent on how servers supply defaults or whether they signal errors for missing commands. All default actions and server-side request modifications SHOULD be reported back to the client.

5.1. Key ? help

This form is required. A server that declares THUMP support MUST respond to the "help" command by listing all the commands (methods) valid for that Key. As a foundation requirement, the "help" command is a common way to ping a THUMP server to see if it is alive.

5.2. Key ? was(DESCRIPTION) when(DATE) resync

This "metadata" command form provides nothing more than a way to carry a Key along with its description. The form is a "no-op" (except when "resync" is present) in the sense that the Key is treated as an adorned URL (as if no THUMP request were present). This form is designed as a passive data structrue that pairs a hyperlink with its metadata so that a formatted description might be surfaced by a client-side trigger event such as a "mouse-over". It is passive in the sense that selecting ("clicking on") the URL should result in ordinary access via the Key-as-pure-link as if no THUMP request were present. The form is effectively a metadata cache, and the DATE of last extraction tells how fresh it is.

The "was" pseudo-command takes multiple arguments separated by "|", the first argument identifying the kind of DESCRIPTION that follows, e.g,

was(erc|Tolstoy, L|War and Peace|1863|http://example.org/etext/2600)

The "when" pseudo-command (optional) takes one argument that is the date that the immediately DESCRIPTION was extracted. The date, conforming to the [TEMPER] specification, looks like YYYYMMDDhhmmss. The "was" and "when" pseudo-commands can harmlessly accompany any THUMP request.

The "resync" command, however, is a request to update the metadata. It returns a "metadata" form similar to the one submitted, but with refreshed metadata and no "resync" at the end.

5.3. Key ? in(DB) find(QUERY) sort([!]ELEMS) list(RANGE) show(ELEMS) as(FORMAT)

This form is used for generalized queries. The server is permitted to modify commands, such as by supplying missing commands (defaults), but SHOULD report the resulting filled-out command xxx.

in(DB)
The "in" command specifies one or more dataset names separated by "|". If no "in" command is present, the server picks a suitable default dataset or returns an error. If no other commands are present, the server may treat the dataset as a result set or return an error. Dataset names originating in relational databases are assumed to name a table in a default database, but may be structured into database, schema, and table names using the reserved characters '/' and '.' as per the following forms:

```
database/schema.table
database/table
schema.table
table
```

find(QUERY)
The "find" command specifies a QUERY that should produce a result set of matching records or an error. The result set is modeled as a numbered sequence of records that is returned "by reference" with a generated Key (see the "results" tag later) or as one or more returned subsequences of records, known as returned sets. If no "find" command is present, Key is expected to imply either a single record or a set of records. THUMP distinguishes between a result set and a returned set, which is a subsequence of the result set included in a given response.

The QUERY consists of free text words separated by spaces. Reserved words begin with a ":" (colon), such as the :and, :or, and :not boolean operators. Parentheses can be used for grouping. Prepending

"+" ("-") to a word is done when the requester desires that the word

be present (absent) from search results. The double-quote character can be used to join words in a phrase or to turn off the special meanings of parentheses or ":+-" in front of words.

sort([!]ELEMS)
The "sort" command is used to request ordering according to the ELEMS specification (descending order if preceded by '!'). If no "sort" command is present, it is up to the server to determine record ordering. ELEMS is one or more element or element subset names separated by "|".

list(RANGE)
The "list" command is used to request that a specific subsequence or RANGE of records be returned. The server should always use the starting point of the requested RANGE, but is free to return fewer records (or a partial record). In all cases the server must report what records or record fragment it has returned. If no "list" command is present, it is up to the server whether to return records, and if so, which records.

RANGE is a pair of arguments, "LENGTH|START", indicating the number of records and starting record in the requested sequence. For example, a RANGE of "10|81" requests 10 records beginning with result set record 81. If both arguments are missing, as in "list()", it is considered a request for all records. If given as just "list(0)", it is a request that no records be returned directly, but a that the result set be returned by reference to a generated Key listed in the "results" tag of the returned set header. If LENGTH is positive and START is 0, the server should send LENGTH randomly selected result set records. If START is missing it defaults to 1; if LENGTH is missing, it is considered a request for all records starting from START.

RANGE may also be used to request record fragments. A returned record set consists of either one or more entire (whole) records, or of exactly one fragment of one record. When a fragment is returned, the start position in the set header (described later) is indicated with S_F, where S is the record number and F is the fragment sequence number. To request the next fragment, a START is formulated by adding 1 to F. For example, "10|45_3" requests 10 records starting at fragment 3 of record 45 (only one fragment can be returned).

show(ELEMS)
The "show" command is used to request that returned records be constituted with ELEMS elements. ELEMS is one or more element or element subset names separated by "|". It can be used by users to

define the composition and element order of a returned record set; element names are discovered by XXX.

Element subset names can also be used. Common subset names are "brief", "full", and "support" (a record that is complete enough to show the server's commitment to the object. If no "show" command is present, it is up to the server which elements to return.

as(FORMAT)
The "as" command is used to request that returned records be formatted according to FORMAT. Common format names are "anvl/erc", "anvl/qdc", and "xml/marc". If no "as" command is present, the default format is usually "anvl/erc" (a plain text format that is eye-readable and machine-readable), although a service may define defaults in its own way.

5.4. Key ?

This is a shorthand for

    Key ? show(brief) as(anvl/erc)

which returns a brief object (identified by Key) description. Support for this shorthand is required.

5.5. Key ??

This is a shorthand for

    Key ? show(support) as(anvl/erc)

which returns an object description full enough to contain the server provider's commitment statement. Support for this shorthand is required.

5.6. Key ? get() put() group() apply()

These commands are currently undefined and reserved by THUMP for future use.

6. Response Summary

A THUMP response consists of a block of HTTP and extension headers, a blank line, and, if the THUMP-Status extension header was 200, a returned set of records. The Content-Type HTTP header is normally returned as

```
    Content-Type: text/plain
```


so that the results will display correctly on a web browser's display. The THUMP content types "text/xml" and "text/html" are being considered.

The rest of this section describes the THUMP extension headers and the structure of the returned record set. Extension headers are inserted in the block of HTTP response headers, usually near the end. Currently, one extension header, THUMP-Status, is defined, and it is required:

```
    THUMP-Status: THUMPVersion StatusCode ReasonPhrase
```

It includes the version, a short human-readable phrase, and a 3-digit integer result code indicating the status of the attempt to execute the request. Defined StatusCodes and ReasonPhrases for THUMP are:

```
    200: OK
    400: Bad Request
    402: Payment Required
    403: Forbidden
    404: Not Found
    405: Method Not Allowed
    408: Request Time-out
```

If the status code is other than 200, no record set should be sent. If the server wishes to convey any more detailed diagnostic or error information than may be expressed by the above status codes, it MUST set the code to 200 and use "error" or "warning" element tags within the returned record set.

A blank line separates the HTTP response and THUMP-Status headers from the returned set that is the body of the response. The returned record set consists of a set-start header record followed by a sequence of records, each separated by one ore more blank lines, until end of stream (file) is reached. A set-end header record is optional.

The format of the records is normally "anvl/erc", which specifies a serialization syntax [ANVL] with ERC semantics [Kernel]. In a future

version of THUMP it will be possible to request ERC semantics with "xml/erc". The next sections describe the special ANVL record used to introduce a record set and then the ERC records.

7. Returned Records

This section describes how a record in the sequence of returned records is encoded in the anvl/erc format. ANVL (A Name Value Language) defines the syntax and the ERC (Electronic Resource

Citation) defines semantics. The URI for the ERC [Kernel] reference should be included in the record set header. While a comprehensive description of the ERC record is out of scope for this document, some details are give below that may suffice for simple implementations.

An ERC record is a sequence of tagged elements. It has the form,

```
erc:
who:   WHO_EXPRESSED_THIS_ITEM
what:  WHAT_THE_EXPRESSION_WAS_CALLED
when:  WHEN_IT_WAS_EXPRESSED
where: WHERE_THE_EXPRESSION_CAN_BE_FOUND
how:   DESCRIPTION_OR_SUMMARY_OF_ITEM              <optional>
why:   COPYRIGHT_DISCLAIMER_AUDIENCE_STATEMENT     <optional>
note:  ANY_TEXT                                    <optional>
       .......
<any other tagged elements>                        <optional>
```

The first five tagged elements are required. The required elements may be thought to answer questions about an "expression" of a resource (an item).

All other elements are optional. The next ERC element shown above ("how") is concerned with the content of an item and the element after that ("why") with any high priority information that comes from the lawyerly domain -- the really hard questions.

A short form of the ERC is also possible that the above ordering for the first 6 elements. It has the form,

```
erc: WHO | WHAT | WHEN
     | WHERE
```

```
          | HOW                                              <optional>
          | WHY                                              <optional>
     note:  ANY_TEXT                                         <optional>
            .......
     <any other tagged elements>                             <optional>
```

The line breaks among the first 6 elements are arbitrary. Together they are considered to be part of one long value for the "erc:" as long as they are continued on indented lines. In either form of the ERC, arbitrary additional elements are possible.

7.1. Empty values for required elements

Although they are required, if no suitable element value can be found, a controlled code value for "empty" of the form

(:ccode)

should be used, drawing from the following reserved values:

(:unac) temporarily inaccessible

(:unal) unallowed, suppressed intentionally

(:unap) not applicable, makes no sense

(:unas) value unassigned (e.g., Untitled)

(:unav) value unavailable, possibly unknown

(:unkn) known to be unknown (e.g., Anonymous, Inconnue)

(:none) never had a value, never will

(:null) explicitly and meaningfully empty

(:tba) to be assigned or announced later

(:etal) too numerous to list (et alia).

(:at) the real value is at the given URL or identifier.

8. FAQ -- Frequently Asked Questions

8.1. What's the difference between THUMP, OpenSearch, SRU/SRW, and OpenURL?

All of these protocols are capable of expressing a parameter package on the right-hand side of a URL, and all of them reserve specific parameter names as having defined meanings. In theory, these packages can be extended arbitrarily to express any functionality with any level of complexity. There's no syntactic limitation to these protocols' expressiveness. The difference lies in how.

THUMP uses a classic parenthesized argument list syntax while the others use the flat argument-value list syntax traditional on the web since 1995. OpenSearch and SRU/SRW are logical descendants of the complex Z39.50 search and retrieve protocol, but with restricted functionality and a text-based syntax. SRW and OpenURL define an XML-encoding for request parameters. OpenURL tends to be used for known-item linking. THUMP aims to be a more concise specification for key-based requests.

9. Security Considerations

The THUMP protocol poses no direct risk to computers and networks. Implementors of THUMP services need to be aware of security issues when querying networks and filesystems, and the concomitant risks from spoofing and obtaining incorrect information. These risks are no greater for THUMP than for any other kind of HTTP-based application. For example, recipients of a URL with embedded THUMP commands should treat it like a URL and be aware that the identified service may no longer be operational.

THUMP clients and servers subject themselves to all the risks that accompany normal operation of the protocols underlying mapping services (e.g., HTTP, Z39.50). As specializations of such protocols, a THUMP service may limit exposure to the usual risks. Indeed, THUMP services may enhance a kind of security by helping users identify long-term reliable references to information objects.

10. References

[ANVL] Kunze, J., Kahle, B., Masanes, J., and G. Mohr, "A Name-Value Language", August 2005, <http://www.cdlib.org/inside/diglib/ark/anvlspec.pdf>.

[ARK] Kunze, J. and R. Rodgers, "The ARK Persistent Identifier Scheme", July 2007, <http://www.cdlib.org/inside/diglib/ark/arkspec.pdf>.

[Kernel] Kunze, J. and A. Turner, "Kernel Metadata and Electronic Resource Citations (ERCs)", October 2007, <http://www.cdlib.org/inside/diglib/ark/ercspec.html>.

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, DOI 10.17487/RFC2822, April 2001, <http://www.rfc-editor.org/info/rfc2822>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC5013] Kunze, J. and T. Baker, "The Dublin Core Metadata Element Set", RFC 5013, DOI 10.17487/RFC5013, August 2007, <http://www.rfc-editor.org/info/rfc5013>.

Authors' Addresses

John Kunze
California Digital Library
415 20th St, #406
Oakland, CA 94612
USA

Email: jak@ucop.edu

Nassib Nassar
Index Data ApS
Njalsgade 76, 13
Copenhagen 2300
Denmark

Email: nassib@indexdata.com