THUMP -- The HTTP URL Mapping Protocol

(http://www.ietf.org/internet-drafts/draft-kunze-thump-02.txt)

Status of this Document

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 kernel metadata as Electronic
Resource Citations, which are similar to blocks of email headers.

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.foo.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.foo.com/object321?show(brief)as(anvl/erc)

An example of a broad database search is,

   http://example.foo.com/?db(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] 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: 0.5 200 OK
      S:
      S: set-start: California Digital Library | THUMP 0.5 | 20060606161407
      S:          | http://ark.cdlib.org/ark:/13030/ft167nb0vq?
  10  S:          | http://dublincore.org/groups/kernel/erc
      S: here: 1 | 1 | 1
      S:
      S: erc:
      S: who:   Stanton A. Glantz and Edith D.  Balbach
  15  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 record set header (lines 8-11) and a single
metadata record (13-17) that comprises the service response.

The record set header identifies (8-11) who created the set, what
created it, when it was created, where an automated process can re-
access the set, and where to look up the meaning of metadata
elements; it ends in a line (11) whose respective sub-elements
indicate that here in this communication the recipient can expect to
find 1 record, starting at the record numbered 1, from a set
consisting of a total of 1 record (i.e., here is the entire set,
consisting of exactly one record).

The returned record (13-17) is in the ERC format. 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's
   conventions might consider

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

   to be equivalent to

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

   There is a natural duality 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 duality 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.

At one extreme of the duality, when the request is completely absent
(no `?' at all), a service may return a THUMP response. This might
make sense for an entire service or only for certain specific Keys.

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.foo.com/?db(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.foo.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://www.gutenberg.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) 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.

The "in" command specifies one or more database names separated by
"|". If no "in" command is present, the server picks a suitable
default database or returns an error. If no other commands are
present, the server may treat the database as a result set or return
an error.

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.

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 "10I81" 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).

The "show" command is used request that returned records be
constituted with ELEMS elements. ELEMS is one or more element or
element subset names separated by "|". Common element 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.

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).

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() sort() 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 THUMPVersion

0.5 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 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 Record Set Header**

What follows is a description of the anvl/erc returned record set
encoding. The first record is a header record of the form,

```
    set-start: WHO_GENERATED | WHAT_SET | WHEN_GENERATED
             | WHERE_TO_RERUN
             | HOW_TO_INTERPRET
    here: NUM_RETURNED_SET | START_POSITION | NUM_RESULT_SET
    results: RESULT_SET_URI                <optional>
    error: FATAL_ERROR_MESSAGE             <optional>
    warning: CAUTIONARY_MESSAGE            <optional>
```

where the upper-cased tokens will be replaced by server-supplied
values. Apart from the "set-start" element, which must appeart
first, the other elements may appear in any order. Just as for an
ordinary record, a blank line ends a set header record.

The last three element tags are optional. If the "error" element is
present, the supplied free-text message indicates a fatal error, and
no usable returned records should be expected. If the "warning"

element is present, the supplied message indicates a non-fatal error,
and usable records may be returned.

[7.1](). "set-start" tag (required)

The "set-start" element must be first in the record. Its associated
values are:

WHO_GENERATED
     The name or URI of the organization generating the set.

WHAT_GENERATED
     The origin of the set, which might be simply "THUMP 0.5" or
perhaps the name of an including collection or database.

WHEN_GENERATED
     The date in [TEMPER] format when the set was generated.

WHERE_TO_RERUN
     A URI containing a THUMP request that can be used to rerun the
originating request (although it need not be a verbatim copy of the
original request), and with differing results not unexpected in cases
where the underlying collection is evolving.

HOW_TO_INTERPRET
     The URI of a document defining the semantics of the element tags
used in the returned record set.

Note that for WHERE_TO_RERUN, the server may mirror the original
THUMP request or may include a revision instead to indicate name
remapping, defaults, interpretations, and corrections. An original
request of

        find(vacuum tube)show(short)

might come back "normalized", for example, as in

        find(vacuum%20tube)list(20|1)show(brief)

[7.2](). "here" tag (required)

The "here" element is required, and has the form,

        here: NUM_RETURNED_SET | START_POSITION | NUM_RESULT_SET

Its associated values are:

NUM_RETURNED_SET
     The number of records in this returned set matching the request.
A 0 (zero) indicates that no records were returned. If the result
set is non-empty, the "results" tag may contain a URI that can be

used as Key in subsequent THUMP requests. The special composite
number, "0_1" indicates a set consisting of exactly one internal

      record fragment, with the final fragment indicated by "1_1". By
      concatenating all the fragments in the correct order, the client can
      reconstruct the whole record.

      START_POSITION
           The starting position of the first returned record relative to
      result set. If this number is 0 (zero), it indicates that the order
      of the returned records is undefined. If this number has the special
      composite form, "S_F", it indicates the record number S and sequence
      number F of the one record fragment in the returned set. The final
      fragment of a record is indicated when the server returns "1_1" for
      NUM_RETURNED_SET.

      Fragments should be requested in sequence by incrementing the
      sequence number, e.g., if 45_2 is returned (fragment 2 of record 45),
      the next fragment request from the client could be "list(10|45_3)",
      which request 10 records (the server unilaterally cuts this down to a
      single fragment) starting from fragment 3 of record 45.

      NUM_RESULT_SET
           The number of records in the result set. If this number is
      followed by a plus (e.g., 345+), it indicates that a minimum number
      that is still subject to growth. If followed by a tilda (e.g.,
      30000~), it indicates an approximate result set size.

      Examples of client/server exchanges that show how "list" commands
      might trigger server responses expressed in "here" returned set
      header tags.

            list(10|1)   -> here: 10 | 1 | 27
            list(10|11)   -> here: 10 | 11 | 27
            list(10|21)   -> here: 7 | 21 | 27

            list(20|1)   -> here: 4 | 1 | 7
            list(20|5)   -> here: 0_1 | 5_1 | 7
            list(20|5_2)   -> here: 0_1 | 5_2 | 7
            list(20|5_3)   -> here: 1_1 | 5_3 | 7
            list(20|6)   -> here: 2 | 6 | 7

7.3. "results" tag (optional)

   The "results" element is optional, and has the form,

         results: RESULT_SET_URI

   Its associated value is a URI that can be used to refer to the
   results in a subsequent THUMP request, e.g.,

```
       RESULT_SET_URI ? list(20|41)
```

   or

         Key ? find(:uri:RESULT_SET_URI :and parklands)

[7.4](#). End of a record set

   The end of a record set is detected when the end of stream (or file)
   is encountered or, optionally, when a record beginning with the "set-
   end" element tag is encountered. This record has the form,

         set-end: OPTIONAL_COMMENT

   The OPTIONAL_COMMENT may contain arbitrary free text and may be
   absent. If a "set-end" is encountered, it is considered to close the
   most recently encountered "set-start". As usual, the record ends
   with a blank line.

[8](#). 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.

8.1. 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:

(:unkn) A null element term explaining that the value is unknown.
         Compared to :unav, this explanation carries a high degree of
         authority regarding the object described. Anonymous
         authorship is an example.

(:unav) A null element term explaining that the value is unavailable
         indefinitely. Compared to :unkn, this explanation is
         intended for intermediary systems that know less about the
         object described and have to rely on the best metadata
         received.

(:unac) A null element term explaining that the value is temporarily
         inaccessible. This might be due, for example, to a system
         outage.

(:unap) A null element term explaining that the value is not
         applicable or makes no sense.

(:unas) A null element term explaining that a value was never
         assigned. An untitled painting is an example.

(:none) A null element term explaining that the element never had a
         value and never will.

(:null) A null element term explaining that the value is explicitly
            empty.

(:unal) A null element term explaining that the value is unallowed
         or suppressed intentionally.

(:tba)   A null element term explaining that the value is to be
         assigned or announced later.

9. FAQ -- Frequently Asked Questions

9.1. 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.

10. Appendix -- Motivation for Electronic Resource Citations (ERCs)

   An Electronic Resource Citation (or ERC, pronounced e-r-c) is a
   simple, compact, and printable record designed to hold data
   associated with an information resource. By design, the ERC is a
   metadata format that balances the needs for expressive power, very
   simple machine processing, and direct human manipulation.

   A founding principle of the ERC is that direct human contact with
   metadata will be a necessary and sufficient condition for the near
   term rapid development of metadata standards, systems, and services.
   Thus the machine-processable ERC format must only minimally strain
   people's ability to read, understand, change, and transmit ERCs
   without their relying on intermediation with specialized software
   tools. The basic ERC needs to be succinct, transparent, and
   trivially parseable by software.

   Borrowing from the data structuring format that underlies the
   successful spread of email and web services, the ERC format uses
   [ANVL], which is based on email and HTTP headers [RFC822]. There is

a naturalness to ANVL's label-colon-value format (seen in the
previous section) that barely needs explanation to a person beginning

   to enter ERC metadata.

   Besides simplicity of ERC system implementation and data entry
   mechanics, ERC semantics (what the record and its constituent parts
   mean) must also be easy to explain. ERC semantics are based on a
   reformulation and extension of the Dublin Core [DCORE] hypothesis,
   which suggests that the fifteen Dublin Core metadata elements have a
   key role to play in cross-domain resource description. The ERC
   design recognizes that the Dublin Core's primary contribution is the
   international, interdisciplinary consensus that identified fifteen
   semantic buckets (element categories), regardless of how they are
   labeled. The ERC then adds a definition for a record and some
   minimal compliance rules. In pursuing the limits of simplicity, the
   ERC design combines and relabels some Dublin Core buckets to isolate
   a tiny kernel (subset) of four elements for basic cross-domain
   resource description.

   For the cross-domain kernel, the ERC uses the four basic elements -
   who, what, when, and where - to pretend that every object in the
   universe can have a uniform minimal description. Each has a name or
   other identifier, a location, some responsible person or party, and a
   date. It doesn't matter what type of object it is, or whether one
   plans to read it, interact with it, smoke it, wear it, or navigate
   it. Of course, this approach is flawed because uniformity of
   description for some object types requires more semantic contortion
   and sacrifice than for others. That is why at the beginning of this
   document, the ARK was said to be suited to objects that accommodate
   reasonably regular electronic description.

   While insisting on uniformity at the most basic level provides
   powerful cross-domain leverage, the semantic sacrifice is great for
   many applications. So the ERC also permits a semantically rich and
   nuanced description to co-exist in a record along with a basic
   description. In that way both sophisticated and naive recipients of
   the record can extract the level of meaning from it that best suits
   their needs and abilities. Key to unlocking the richer description
   is a controlled vocabulary of ERC record types (not explained in this
   document) that permit knowledgeable recipients to apply defined sets
   of additional assumptions to the record.

10.1. **ERC Syntax**

   An ERC record is a sequence of metadata elements ending in a blank
   line. An element consists of a label, a colon, and an optional
   value. Here is an example of a record with five elements.

```
        erc:
        who: Gibbon, Edward
        what: The Decline and Fall of the Roman Empire
        when: 1781
        where: http://www.ccel.org/g/gibbon/decline/
```

A long value may be folded (continued) onto the next line by
inserting a newline and indenting the next line. A value can be thus
folded across multiple lines. Here are two example elements, each
folded across four lines.

```
        who/created: University of California, San Francisco, AIDS
              Program at San Francisco General Hospital | University
              of California, San Francisco, Center for AIDS Prevention
              Studies
        what/Topic:
               Heart Attack | Heart Failure
              | Heart
                              Diseases
```

An element value folded across several lines is treated as if the
lines were joined together on one long line. For example, the second
element from the previous example is considered equivalent to

```
        what/Topic: Heart Attack | Heart Failure | Heart Diseases
```

An element value may contain multiple values, each one separated from
the next by a `|' (pipe) character. The element from the previous
example contains three values.

For annotation purposes, any line beginning with a `#' (hash)
character is treated as if it were not present; this is a "comment"
line (a feature not available in email or HTTP headers). For
example, the following element is spread across four lines and
contains two values:

```
        what/Topic:
              Heart Attack
        #    | Heart Failure  -- hold off until next review cycle
              | Heart Diseases
```

11. 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.

12. Authors' Addresses

Kevin Gamiel
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
Duke University
North Carolina State University

Fax: +1 919-445-9669
EMail: kgamiel@renci.org

John A. Kunze
California Digital Library
University of California, Office of the President
415 20th St, 4th Floor
Oakland, CA 94612-3550, USA

Fax: +1 510-893-5212
EMail: jak@ucop.edu

Nassib Nassar
Etymon Systems
P.O. Box 12484
Research Triangle Park, NC 27709, USA

EMail: nassar@etymon.com

13. Informative References

[ANVL]   J. Kunze, B. Kahle, et al, "A Name-Value Language", work
           in progress,
           http://www.cdlib.org/inside/diglib/ark/anvlspec.pdf

[ARK]   J. Kunze, "Towards Electronic Persistence Using ARK
           Identifiers", Proceedings of the 3rd ECDL Workshop on Web
           Archives, August 2003, (PDF)
           http://bibnum.bnf.fr/ecdl/2003/proceedings.php?f=kunze

[DCORE]   Dublin Core Metadata Initiative, "DCMI Terms", December

2006, http://dublincore.org/documents/dcmi-terms/.

   [KERNEL]   Dublin Kernel Metadata, work in progress within the Dublin
               Core Metadata Initiative (DCMI) Kernel Working Group,
               http://dublincore.org/groups/kernel/

   [RFC822]   D. Crocker, "Format of ARPA Internet Text Messages",
               August 1982, http://www.ietf.org/rfc/rfc822.txt

   [RFC3629]   F. Yergeau, "UTF-8, a Transformation Format of ISO 10646",
               November 2003, http://www.ietf.org/rfc/rfc3629.txt

14. Copyright Notice

Table of Contents