Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 12, 2025

J. Michaud

February 8, 2025

Hypertext Command Line Interface
draft-michaud-hcli-00

Abstract

This document proposes a codification of resources and their relations with hyperlinks, using Unix-like command line interface (CLI) semantics, to foster the creation of a reusable and prolific intersection between the REST architectural style and the pipe and filter style.

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."

Table of Contents

1. Introduction

The proliferation of hypermedia-rich media-types such as HAL, Collection+JSON, Siren, JSON API, JSON-LD, Mason, etc., was a significant step in helping hypermedia APIs take flight in the industry. However, given the generic nature of these hypermedia-rich media types, the establishment of meaning within their structure requires additional work. Semantic definition and agreement between API providers and consumers is required to enable use coherence to emerge.

Hypertext Command Line Interface (HCLI) is a standard that establishes semantic conventions for expressing command line interfaces via hypermedia controls, such as through the use of links and forms.

HCLI leverages the use of the Application Level Profile Semantics (ALPS) [I-D.draft-amundsen-richardson-foster-alps], similar to microformats, to establish a shared understanding and meaning within generic hypermedia-rich media-types.

HCLI's conventions result in a semantic interface, a connector, for serving and consuming CLIs, enabling for the creation of general-purpose libraries and clients that can be re-used. The establishment of a semantic interface of this nature lays down a foundation for the instigation of a rich ecosystem of reusable and shareable CLIs.

The primary design goal of HCLI is to foster the creation of a reusable and prolific intersection, or bridge, between the REST architectural style and the pipe and filter style; as the latter is seen under the Unix/Linux shell. HCLI can be applied to any domain, and only imposes limitations that constrain an HTTP API to operate as a hypertext command line interface.

2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. HCLI ALPS Semantics

The HCLI semantics are largely codified via the definition of a proposed standard ALPS profile [I-D.draft-amundsen-richardson-foster-alps], and takes the following form:

```
<alps version="1.0">
  <doc>Hypertext Command Line Interface (HCLI) semantics.</doc>

  <descriptor id="hcli-document" name="cli">
    <doc>A resource for providing HCLI argument choices</doc>

    <descriptor rt="#command" type="safe" name="cli"/>
    <descriptor rt="#option" type="safe" name="cli"/>
    <descriptor rt="#parameter" type="safe" name="cli"/>
    <descriptor rt="#execution" type="safe" name="cli"/>

    <descriptor href="#hcli_version" />
    <descriptor href="#name" />
    <descriptor href="#section" />
  </descriptor>

  <descriptor id="command" type="semantic">
    <doc>A resource for describing and selecting a command.</doc>

    <descriptor rt="#hcli-document" name="cli" type="safe"/>

    <descriptor href="#hcli_version"/>
    <descriptor href="#name" />
    <descriptor href="#description" />
  </descriptor>

  <descriptor id="option" type="semantic">
    <doc>A resource for describing and selecting an option.</doc>

    <descriptor rt="#hcli-document" name="cli" type="safe"/>

    <descriptor href="#hcli_version"/>
    <descriptor href="#name" />
    <descriptor href="#description" />
  </descriptor>

  <descriptor id="parameter" type="semantic">
    <doc>A resource for providing an input parameter.</doc>

    <descriptor rt="#hcli-document" name="cli" type="safe">
      <doc>A form to input a command line parameter.</doc>
```

```
        <descriptor id="hcli_param" type="semantic"/>
      </descriptor>

      <descriptor href="#hcli_version"/>
    </descriptor>

    <descriptor id="execution" type="semantic">
      <doc>A resource for executing a command line.</doc>

      <descriptor name="cli" type="safe" />
      <descriptor name="cli" type="unsafe" />
      <descriptor name="cli" type="idempotent" />

      <descriptor href="#hcli_version"/>
      <descriptor href="#command"/>
      <descriptor href="#http"/>
    </descriptor>

    <descriptor id="hcli_version" type="semantic">
      <doc>The version of the HCLI specification in use</doc>
    </descriptor>

    <descriptor id="name" type="semantic">
      <doc>The name of the described command or option</doc>
    </descriptor>

    <descriptor id="description" type="semantic">
      <doc>The description of a named section, command, or option</doc>
    </descriptor>

    <descriptor id="section" type="semantic">
      <doc>A man page style HCLI documentation section.</doc>

      <descriptor href="#name" />
      <descriptor href="#description" />
    </descriptor>

    <descriptor id="method" type="semantic">
      <doc>A protocol hint of the default method to use when executing an HCLI.</doc>
    </descriptor>
  </alps>
```

4. HCLI Document

An HCLI Document, or any document containing HCLI semantics, MAY be represented using any media-type that is able to express the HCLI semantics described via the proposed ALPS profile and the provisions otherwise mentioned in this specification.

It should also be noted that an HCLI Document MUST NOT endeavor to expose the entirety of a command line interface in a single HCLI Document. HCLI serving MUST be represented incrementally and across multiple HCLI Documents as each layer of commands, options and parameters unfold through linked navigation.

An HCLI Document MUST always contain at least one of the following:

(1) Command

(2) Option

(3) Parameter

(4) Execution

The "hcli_" prefix is reserved by the HCLI specification and MUST NOT be used by API creators to define private semantics.

This specification makes use of one reserved property and one reserved URI Template [RFC6570] Parameter name, using the "hcli_" prefix:

(1) "hcli_version": The version of this HCLI specification.

(2) "hcli_param": The Parameter name to submit in a URI Template.

Additionally, this specification reserves the following names for exclusive use by HCLI clients:

(1) "options" : A reserved section name

(2) "commands" : A reserved section name

(3) "help": A reserved Command name

4.1. Reserved Property and Other Reserved Names

4.1.1. hcli_version

The reserved "hcli_version" property is REQUIRED.

This property MUST be present in all HCLI Resources and MUST contain the version number of the applicable HCLI specification.

This specification documents the semantics of HCLI version "1.0".

4.1.2. Other Reserved Names

Other reserved names, mentioned in [Section 4. HCLI Document], and related provisions are covered in sections further below, where they can be better contextualized.

4.2. name

The "name" property is REQUIRED.

The name property contains the name of the Command described by the HCLI Document.

4.3. section

The "section" property is REQUIRED.

A section contains documentation used by users to interact with an HCLI. Given the primary goal of HCLI, mentioned in the introduction, the sections SHOULD take a form akin to Unix/Linux style man pages and MUST minimally make use of the following named sections:

(1) name

(2) synopsis

(3) description

(4) examples

The purpose and content of each section SHOULD adhere to Unix/Linux man page conventions and the first three sections MUST be in the exact sequence mentioned above. Other sections MAY be in any order.

Two section names are reserved for automatic documentation generation by HCLI clients and MUST NOT be used in HCLI Documents:

(1) options

(2) commands

4.3.1. name

The "name" property is REQUIRED.

Its value SHOULD be the name of a documentation section.

4.3.2. description

The "description" property is REQUIRED.

Its value SHOULD contain information that pertains to the purpose of the named section.

4.4. Command

Command is OPTIONAL.

A Command describes a type of action to effect in a command line execution.

A Command MUST only be made available if contextually relevant, for a given HCLI Document, in the cumulative formation of a command line sequence. Specific contextual relevance determination is, however, out of scope of this specification.

Because many Commands and Options MAY be presented in an HCLI Document, individual Command "cli" links presented in an HCLI Document SHOULD be disambiguated, if possible, by using a secondary key containing the name of the Command.

Successful traversal, by an HCLI client, of a the "cli" link relation in a Command, SHOULD return an HCLI Document describing the targeted Command’s context.

4.4.1. name

The "name" property is REQUIRED.

Its value SHOULD be the name of the Command (e.g. "s3" in "aws s3").

By convention, and to help human actors differentiate Options from Commands, Command names SHOULD NOT be prefixed with a:

(1) single dash character (i.e. "-")

(2) two consecutive dash characters (i.e. "--").

Furthermore, to make it easier for HCLI clients to differentiate between Parameters, Commands and Options, the name value SHOULD NOT make use of the:

(1) single quote character (i.e. ’)

(2) double quote character (i.e. ")

(3) back tick character (i.e. ‘)

The "help" Command name is reserved for use by clients and MUST NOT be presented as a valid Command to use.

4.4.2. description

The "description" property is REQUIRED.

Its value SHOULD contain information that describes the purpose of the Command.

4.5. Option

Option is OPTIONAL.

An Option describes an option or flag that MAY be used to effect a variation on the outcome of a command line execution.

An Option MUST only be made available if contextually relevant, for a given HCLI Document, in the cumulative formation of a command line sequence. Specific contextual relevance determination is, however, out of scope of this specification.

Successful traversal, by an HCLI client, of a "cli" link relation in an Option SHOULD return an HCLI Document that correlates to the last successfully navigated Command. Options MUST NOT present documentation distinct from that of it’s associated Command.

Because many Commands and Options MAY be presented in an HCLI Document, individual Option "cli" links presented in an HCLI Document SHOULD be disambiguated, if possible, by using a secondary key containing the name of the Option.

The "cli" link relation contained in an Option that has already been navigated through by an HCLI client MUST NOT be presented any longer. The Option itself, however, MUST otherwise remain available in an HCLI Document to help HCLI clients with automatic documentation generation.

4.5.1. name

The "name" property is REQUIRED.

By convention, and to help human actors differentiate between Options

and Commands, Option names SHOULD be prefixed with a:

(1) single dash character (i.e. "-")

(2) two consecutive dash characters (i.e. "--").

To make it easier for clients to differentiate between Parameters, Commands and Options, the name value SHOULD NOT make use of the:

(1) single quote character (i.e. ’)

(2) double quote character (i.e. ")

(3) back tick character (i.e. ‘)

4.5.2. description

The "description" property is REQUIRED.

Its value SHOULD contain information that describes the purpose of the Option, and how to use it.

4.6. Parameter

Parameter is OPTIONAL.

A Parameter describes an HCLI client’s ability to input a single user supplied parameter, using the reserved "hcli_param" key. The "hcli_param" key is used as a simple string expansion variable in a URI Template (see section 3.2.2 of [RFC6570]), that MAY be required for successful command line execution.

A submitted expansion variable MUST be URL encoded [RFC3986].

A Parameter MUST only be made available if contextually relevant, for a given HCLI Document, in the cumulative formation of a command line sequence. Specific contextual relevance determination is, however, out of scope of this specification.

Successful traversal, by an HCLI client, of the "cli" link relation contained in a Parameter, SHOULD return an HCLI Document that correlates to the last successfully navigated Command. Parameters MUST NOT present documentation distinct from that of it’s associated Command.

An HCLI Document MUST NOT contain more than a single Parameter at a time but more than one Parameter MAY be presented sequentially across HCLI Documents.

4.7. Execution

Execution is OPTIONAL.

Execution identifies an HCLI client’s ability to execute a theretofore accumulated command line.

Execution MUST only be made available if contextually relevant, for a given HCLI Document and an accumulated command line sequence. Specific contextual relevance determination is, however, out of scope of this specification.

Successful traversal, by an HCLI client, of the "cli" link relation contained in Execution, MUST NOT return an HCLI Document and effectively steps out of scope of HCLI semantics. API provider semantics are also out of scope of this specification.

4.7.1 command

The command property is REQUIRED.

The command property MUST contain a summary of the theretofore accumulated command line sequence.

4.7.2 http

The http property is REQUIRED.

The http property MUST provides a protocol hint of the default HTTP method to automatically use by HCLI clients when navigating through the final "cli" link relation leading into API provider specific semantics.

Note that only a single default value is allowed. The default value MUST align with one of the allowed methods for the resource referenced through the "cli" link relation.

4.7.2.1 HTTP Protocol

For the HTTP protocol, the allowed method values are:

(1) for a safe "cli" link transition

- get

(2) for an unsafe, idempotent "cli" link transition

- put

- delete

(3) for an unsafe, non-idempotent "cli" link transition

- post

- patch

5. Examples

5.1 usp5

Here’s an example of a hypothetical "usp5" hypertext command line interface represented as hal+json [I-D.draft-kelly-json-hal] and navigating us through a request for the CLI’s version number:

```
GET /usp5 HTTP/1.1
Host: example.org
Accept: application/hal+json

HTTP/1.1 200 OK
Content-Type: application/hal+json

{
  "_links": {
    "self": {
      "href": "/usp5"
    },
    "profile": {
      "href": "http://example.org/profiles/hcli#hcli-document"
    },
    "cli": [
      {
        "href": "/usp5/__cdef/admin?command=usp5",
        "profile": "href": "http://example.org/profiles/hcli#command"
        "name": "admin"
      },
      {
        "href": "/usp5/__cdef/admin?command=usp5+--version",
        "profile": "href": "http://example.org/profiles/hcli#option"
        "name": "--version"
      }
    ]
  },
  "hcli_version": "1.0",
  "name": "usp5",
  "section" : [
```

```
     {
       "name": "name",
       "description": "usp5"
     },
     {
       "name": "synopsis",
       "description": "usp5 [option] <command> <subcommand> [parameters]"
     },
     {
       "name": "description",
       "description": "The usp5 CLI is a tool used to manipulate udp session manager prot
ocol (usp5) users and credentials."
     },
     {
       "name": "examples",
       "description": "N/A"
     }]
   }

   GET /usp5/__odef/--version?command=usp5 HTTP/1.1
   Host: example.org
   Accept: application/hal+json

   HTTP/1.1 200 OK
   Content-Type: application/hal+json
   {
     "_links": {
       "self": {
         "href": "/usp5/__odef/--version?command=usp5"
       },
       "profile": {
         "href": "http://example.org/profiles/hcli#option"
       },
       "cli": {
         "href": "/usp5?command=usp5+--version",
         "profile": "http://example.org/profiles/hcli#hcli-document"
       }
     },
     "hcli_version": "1.0",
     "name": "--version",
     "description": "The usp5 CLI version."
   }

   GET /usp5?command=usp5+--version HTTP/1.1
   Host: example.org
   Accept: application/hal+json

   HTTP/1.1 200 OK
   Content-Type: application/hal+json
```

```
   {
     "_links": {
       "self": {
         "href": "/usp5?command=usp5+--version"
       },
       "profile": {
         "href": "http://example.org/profiles/hcli#hcli-document"
       },
       "cli": [
         {
           "href": "/usp5/__cdef/admin?command=usp5+--version",
           "profile": "href": "http://example.org/profiles/hcli#command"
           "name": "admin"
         },
         {
           "href": "/usp5/__cdef/admin?command=usp5+--version",
           "profile": "href": "http://example.org/profiles/hcli#option"
           "name": "--version"
         },
         {
           "href": "/usp5/__edef?command=usp5+--version",
           "profile": "href": "http://example.org/profiles/hcli#execution"
         }
       ]
     },
     "hcli_version": "1.0"
     "name": "--version",
     "section" : [
     {
       "name": "name",
       "description": "usp5"
     },
     {
       "name": "synopsis",
       "description": "usp5 [option] <command> <subcommand> [parameters]"
     },
     {
       "name": "description",
       "description": "The usp5 CLI is a tool used to manipulate udp session manager prot
ocol (usp5) users and credentials."
     },
     {
       "name": "examples",
       "description": "N/A"
     }]
   }

   GET /usp5/__edef?command=usp5+--version HTTP/1.1
   Host: example.org
```

```
Accept: application/hal+json

HTTP/1.1 200 OK
Content-Type: application/hal+json

{
  "_links": {
    "self": {
      "href": "/usp5/__edef?command=usp5+--version"
    },
    "profile": {
      "href": "http://example.org/profiles/hcli#execution"
    },
    "cli": {
      "href": "/usp5/execution?command=usp5+--version",
    }
  },
  "hcli_version": "1.0",
  "command": "usp5 --version",
  "http": "get"
}

GET /usp5/execution?command=usp5+--version HTTP/1.1
Host: example.org
Accept: application/octet-stream

HTTP/1.1 200 OK
Content-Type: application/octet-stream

0.0.1
```

5.2 jsonf

Here’s an example of a hypothetical "jsonf" hypertext command line interface represented as hal+json [I-D.draft-kelly-json-hal] that takes a JSON input stream and attempts to reformat it to enhance human readability:

```
GET /jsonf HTTP/1.1
Host: example.org
Accept: application/hal+json

HTTP/1.1 200 OK
Content-Type: application/hal+json

{
  "_links": {
    "self": {
```

```
        "href": "/jsonf?command=jsonf"
      },
      "profile": {
        "href": "http://example.org/profiles/hcli#hcli-document"
      },
      "cli": [
        {
          "href": "/jsonf/__edef?command=jsonf",
          "profile": "href": "http://example.org/profiles/hcli#execution"
        }
      ]
    },
    "hcli_version": "1.0"
    "name": "jsonf",
    "section" : [
    {
      "name": "name",
      "description": "jsonf"
    },
    {
      "name": "synopsis",
      "description": "jsonf"
    },
    {
      "name": "description",
      "description": "The jsonf CLI formats JSON documents."
    },
    {
      "name": "examples",
      "description": "cat linear.json | jsonf"
    }]
  }

  GET /jsonf/__edef?command=jsonf HTTP/1.1
  Host: example.org
  Accept: application/hal+json

  HTTP/1.1 200 OK
  Content-Type: application/hal+json

  {
    "_links": {
      "self": {
        "href": "/jsonf/__edef?command=jsonf"
      },
      "profile": {
        "href": "http://example.org/profiles/hcli#execution"
      },
```

```
      "cli": {
        "href": "/jsonf/execution?command=jsonf",
      }
    },
    "hcli_version": "1.0",
    "command": "jsonf go",
    "http": "post"
  }

  POST /jsonf/execution?command=jsonf HTTP/1.1
  Host: example.org
  Content-Type: application/json
  Accept: */*

  {"linear":"and","hard":"to","read":"json"}

  HTTP/1.1 200 OK
  Content-Type: application/json

  {
    "linear" : "and",
    "hard" : "to",
    "read" : "json"
  }
```

6. Stream Processing

6.1.1. For Servers

Servers implementing HCLI semantics that leverage unsafe Execution (HTTP POST or HTTP PATCH) and that intend to allow input stream-like processing for their HCLI, for example, by receiving an STDIN input stream through a filter from a pipe and filter sequence in a Unix/Linux shell, MUST minimally be ready to receive and send entity bodies of the application/octet-stream media-type [RFC2046].

This is intended to easily enable the pipe and filter style for clients and servers that understand and offer HCLI semantics. Specific input stream processing semantics by services are out of scope of this specification.

Servers SHOULD, however, in as much as it doesn’t interfere with pipe and filter streaming behavior, endeavor to support receiving and sending correctly identified media-types to help maintain visibility on the network.

Servers intending for clients to present coherent error messaging to

the standard error output stream (STDERR) SHOULD return an HTTP API problem detail [RFC9457].

Servers SHOULD abide by the Unix philosophy regarding successes. Notably, servers SHOULD favor "success equals silence" to enable piping without noise from success messages, and CLI behavior SHOULD remain predictable and composable.

6.1.2. For Clients in General

Command line clients that support input stream processing SHOULD attempt to send their input stream to an HCLI server, but only if a command line sequence leads to an unsafe HTTP POST or idempotent HTTP PUT Execution.

Clients MUST minimally support sending input streams (STDIN) and receiving output streams (STDOUT) under the application/octet-stream [RFC2046] media-type.

Command line clients SHOULD, however, endeavor to communicate input streams by using the most appropriate media-type representing the input stream to transfer, and SHOULD also be ready to process responses making use of the most appropriate media-type for the received response, to help maintain visibility on the network.

Clients SHOULD also be ready to support HTTP API problem details [RFC9457] to help yield a sensible standard error output stream (STDERR).

Clients SHOULD give navigation precedence to a Parameter, over Commands and Options, when presented in an HCLI Document.

6.1.2. For Command Line Clients

Command line interface (CLI) clients SHOULD support receiving input from STDIN, or other equivalent, to allow for pipe and filter processing.

Command line clients receiving the result of a successful HCLI command line execution SHOULD output the result to STDOUT, or other equivalent, to allow for further pipe and filter processing.

Command line clients receiving failure results from an HCLI command line execution in the form of an HTTP API problem detail [RFC9457] SHOULD output the failure result detail to STDERR, or other equivalent, to allow for further pipe and filter processing.

7. Recommendations

7.1. Standard HCLI ALPS Profile

The standard HCLI ALPS profiles defined in this specification MAY be hosted alongside an HCLI API deployment and correspondingly referenced in an HCLI Document to enable self-contained deployment. HCLI Documents SHOULD, however, reference the standard HCLI ALPS profile hosted at an independent ALPS registry (APR).

7.2. For Clients in General

7.2.1. Command Line Execution

Submitting a command line sequence SHOULD prompt a client to automatically, and sequentially parse submitted arguments, and to navigate through sequences of HCLI Documents, by using Command or Option "name" or the Parameter variable "hcli_param", corresponding to each parsed command, option or parameter.

A client MAY want to facilitate disambiguation of user supplied parameters from commands and options by always assuming that strings surrounded by the double quote character (i.e. "), or the single quote character (i.e. ’), are intended to be submitted as a user supplied parameter.

Upon running out of commands, options and parameters to parse for use in HCLI navigation, a client SHOULD automatically attempt an Execution. If Execution isn’t available, the command line execution SHOULD be treated as a failure and the last successfully retrieved HCLI Document MAY be used to provide useful feedback to help users correct mistakes.

7.2.2. Documentation

The reserved "help" command MAY be used anywhere in a command line sequence and SHOULD present documentation formatted from the last successfully retrieved HCLI Document preceding the "help" command.

7.2.3. Robustness

To maximize robustness, clients SHOULD endeavor to follow Postel’s law (e.g. be conservative in what you do, be liberal in what you accept from others).

In other words, and of particular interest in this case, clients consuming coherent HCLI semantics intermixed with non-HCLI semantics they don’t understand, SHOULD favor ignoring what they don’t understand over failing.

7.3. For Command Line Clients (e.g. Under Unix/Linux shell)

7.3.1. Documentation

When generating documentation via "help", a man page SHOULD be presented and formatted from the last successfully retrieved HCLI Document preceding the "help" command.

If man pages are not available for use in the command line environment, an environment specific equivalent SHOULD be used to provide users with useful documentation.

Additionally, clients SHOULD automatically generate an OPTIONS and COMMANDS section, two sections reserved for client use, that corresponds to a listing of Options and Commands referenced in the presented HCLI Document.

8. Security Considerations

TBD

9. IANA Considerations

This specification introduces the "cli" link relation per [RFC5988] to indicate the presence of command line interface semantics.

The link relation SHOULD be accompanied by a "profile" [RFC6906] defining command line interface semantics.

This specification defines a standard ALPS profile to be used in conjunction with the "cli" link relation but does NOT REQUIRE that the HCLI profile defined in this specification be the only profile used in conjunction with the "cli" link relation.

10. References

10.1. Normative References

[RFC6906] Wilde, E., "The ’profile’ Link Relation Type", RFC 6906, March 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC

3986, January 2005.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

[RFC2046] Freed, N., Innosoft, Borenstein, N., First Virtual, "Multipurpose Internet Mail Extensions", RFC 2046, November 1996.

[RFC9457] Nottingham, M., Wilde, E., Dalal, S., "Problem Details for HTTP APIs", RFC 9457, March 2016.

10.2. Informative References

[I-D.draft-kelly-json-hal]

Kelly, M., "JSON Hypertext Application Language", draft-kelly-json-hal-11 (work in progress), October 2023.

[I-D.draft-amundsen-richardson-foster-alps]

Amundsen, M., CA Technologies, Inc., Richardson L., Foster, M., "Application-Level Profile Semantics (ALPS)", draft-amundsen-richardson-foster-alps-04 (work in progress), October 2020.

Appendix A. Acknowledgements

Thanks to Mike Amundsen, Mark Foster and others on the ALPS Discussion list for providing invaluable feedback during the definition of the HCLI ALPS profile.

Appendix B. Frequently Asked Questions

B.1. What are some of the benefits of HCLI?

B.1.1. Reduced Cost

The most immediate benefit is that instead of having to develop a custom client for your HCLI API, you can benefit from the use of a standard client that understands HCLI semantics. Such a client would be able to interact out-of-the-box with any HCLI as an imposter CLI; you get a free bridge into the pipe and filter style for your HCLI connector fronted API.

B.1.2. Self-documenting and Up-to-date

An HCLI is self-documenting in that the commands, options and parameters that can be selected are plainly visible through the documentation mechanisms highlighted in this specification. Moreover, given the distributed nature of the HCLI semantic interface, changes made to an HCLI are always distributed to all clients using the API. Clients can always benefits from up-to-date functionality and up-to-date documentation.

B.1.3. Immediacy

Having at your disposal an instantly usable shell CLI for your HCLI provides developers with an immediately usable, and familiar, solution through a familiar mode of operation (pipe and filter).

B.1.4. Scripting and Experimentation

By enabling the pipe and filter style, HCLI also enables notions of scripting, an inherent and well understood companion of the pipe and filter style under shell (e.g. Unix/Linux). HCLI enables scripting orchestration and experimentation.

B.1.5. Building Blocks

HCLI also raises notions of reusable HCLIs and of an ecosystem of shareable and reusable HCLIs to be used as building blocks to help in the construction and orchestration of APIs, with minimal work.

B.1.5. Native

Because the entire HCLI semantic interface (connector) leading all the way up to an API surface (component) is safe from side-effects, we raise the possibility of "nativizing" an HCLI by allowing for a CLI to be auto-generated, on demand, to run locally and natively

(without the extra overhead of navigating through hypertext);
improving on performance and creating a "frozen" version of a CLI.

Author’s Address

Jeff Michaud

Email: cometaj2@proton.me
Twitter: @cometaj2