Internet Draft
<draft-booth-clip-00.txt>
Creation Date: 2011-01-20
Category: Experimental
Expires July 2011

B. Booth
Motorola Mobility

January 2011

Common Logic Interface Protocol (CLIP) Framework
<draft-booth-clip-00.txt>

Status of this Memo

Distribution of this memo is unlimited.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on July 24, 2011.

Copyright Notice

Abstract

This document defines the Common Logic Interface Protocol (CLIP) framework. CLIP provides a programming language independent interface between system components. An overview of CLIP is provided as well as a description of logical systems. In addition, the CLIP syntax is defined. Example pseudo-code for common clip utility functions are provided. Finally, some additional notes and recommended practices are included in the document.

Table of Contents

1. Introduction

1.1. Overview

The Common Logic Interface Protocol (CLIP) framework provides independent logical systems a simple means of connecting and inter-operating. CLIP provides a platform and programming language-independent interface between system components. CLIP's roots are primarily derived from data flow design methodology and the fundamental common gateway interface (CGI) [RFC3875] concept of name/value pair queries typically found on web server URI [RFC3986] implementations. CLIP is designed to be scaleable. Specifically, CLIP is intended to be a practical solution that is applicable from small embedded devices to large scale server environments. In practice, a CLIP system may be implemented in a multitude of environments. For example, a script running on a web server, code running on a client side browser, a native application running on a personal computer, or firmware within an embedded device. Additionally, systems that utilize CLIP can interconnect via any data communication method (e.g. TCP/IP sockets, HTTP, data queues, device memory interface, software programming interfaces, etc.). It should be noted that CLIP messages are intended to be REST compatible.

1.2. 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 RFC 2119 [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the 'must' requirements for the protocols it implements. An implementation that satisfies all of the 'must' and all of the 'should' requirements for its features is said to be 'unconditionally compliant'; one that satisfies all of the 'must' requirements but not all of the 'should' requirements for its features is said to be 'conditionally compliant'.

2. Logical Systems

A logical system is one or more expressions or sub-systems that operate on a set of input parameters and MAY generate one or more output parameters. The input and output to and from logical systems are name/value pairs as specified in the CLIP Syntax [section 3.1].

Figure 1 shows a high level overview of a simple logical system.

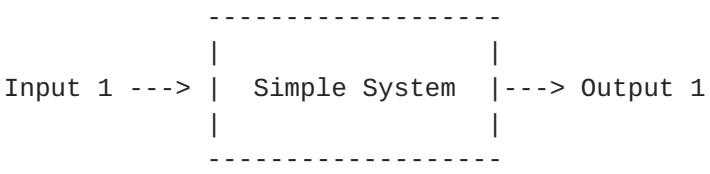


Figure 1 - Simple CLIP Logical System

In practice, a logical system will contain five fundamental elements: a data input receiver (data_recv), an input handler (clip_in), a logical system (system), an output generator (clip_out), and a data output sender (data_send). Figure 1.1 shows these internal elements using the Simple System from Figure 1 as a model.

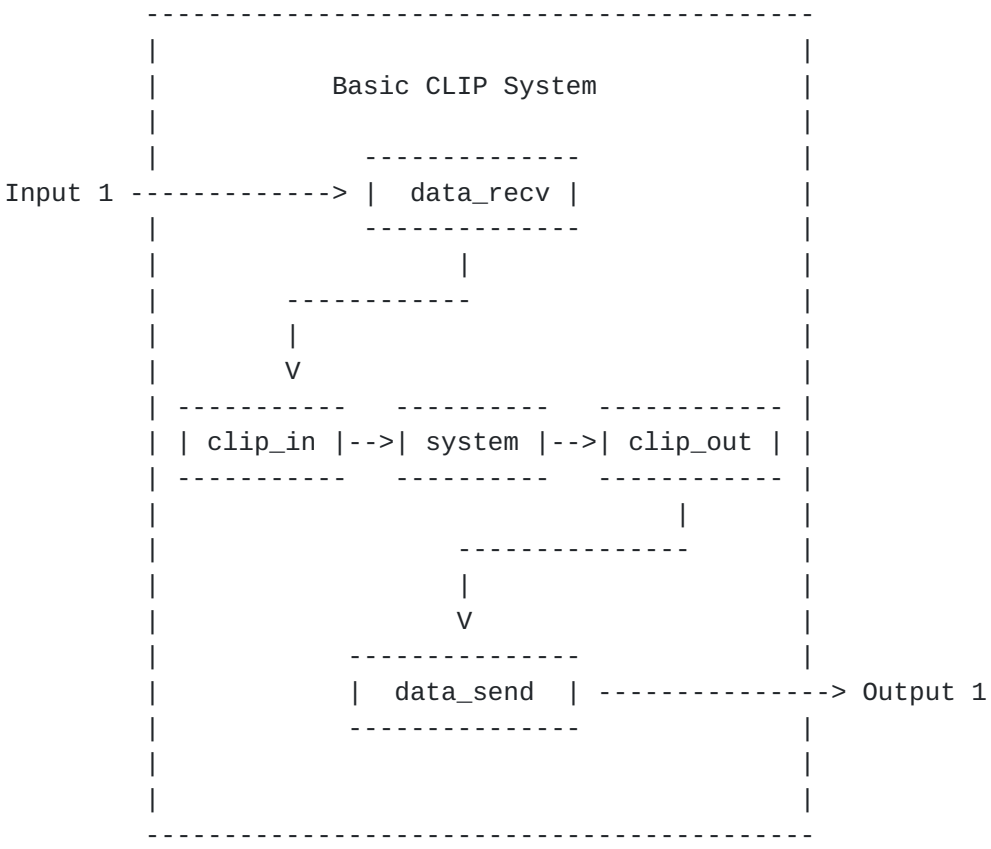


Figure 1.1 - CLIP Logical System Internal Elements

The input receiver (data_recv) is responsible for capturing input data from an input source. The input handler (clip_in) parses input name/ value strings into a system dependent native format. The system operates on the native input name/value pairs. The output generator (clip_out) builds output name/value strings from the system's native format. The output sender (data_send) distributes the output data to its intended destination.

Figure 2 shows a complex logical system that includes multiple input and output parameters.

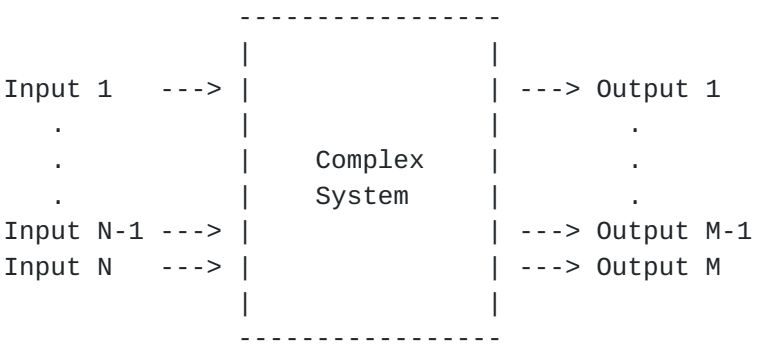


Figure 2 - Complex CLIP Logical System

CLIP can be utilized in both simple and complex logical systems. Systems can be inter-"clipped", intra-"clipped", or any combination thereof.

Figure 3 shows two systems that are inter-clipped. In this case, Output 1 from System A is utilized as Input 2 into System B. Correspondingly, Output 2 from System B is channeled to System A as Input 1.

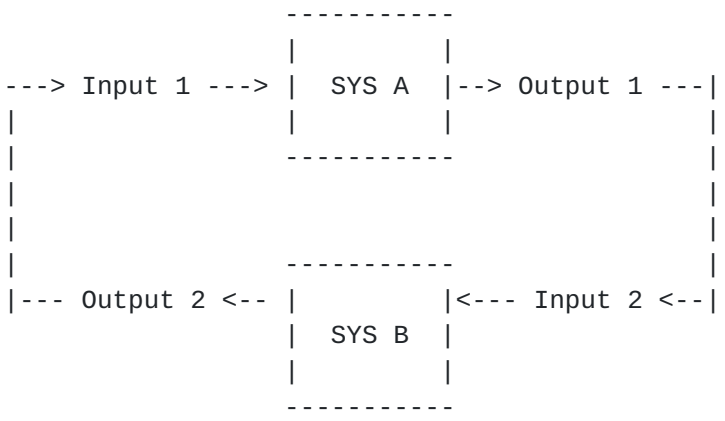


Figure 3 - Inter-clipped Logical Systems

Figure 4 shows two logical systems that are intra-clipped. In this case, Input A1 of System A is utilized as Input B1 into System B. Correspondingly, Output B2 from System B is redirected as the Output A2 from System A. In this example, System A also has an additional Output A1.

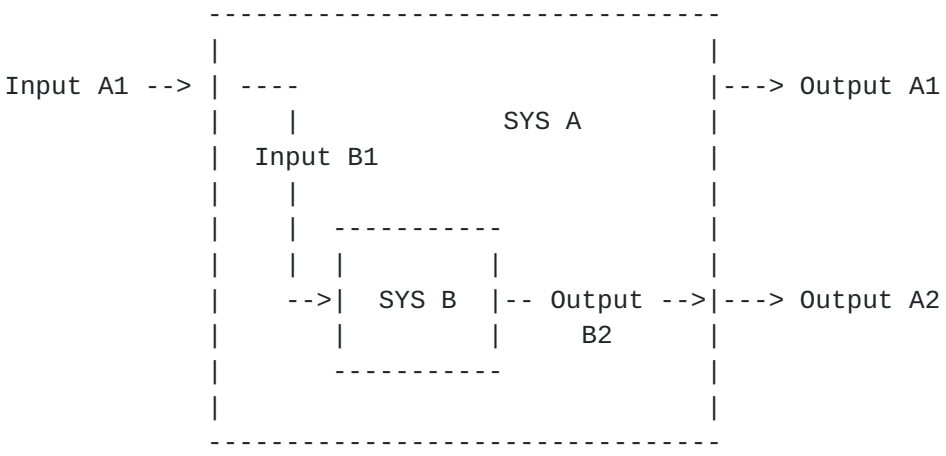


Figure 4 - Intra-clipped Logical Systems

3. CLIP

3.1 Syntax

The CLIP Interface provides a common format for input and output to and from logical systems and is specified as a set of name value pairs delimited by the ampersand ("&") character. Each name/value pair in a set is delimited by the equals ("=") character.

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

```
uchar          = any unicode character except for
                 "&", "=", or "%"

nv_char        = uchar / "%25" / "%26" / "%3D"

name           = *( nv_char )

value          = *( nv_char )

nv_pair        = *( [ name "=" ] value )

clip_input     = *( nv_pair *( "&" nv_pair ) )

clip_output    = *( nv_pair *( "&" nv_pair ) )
```

3.2 Pseudo Code

The code examples (written in Javascript below) outline the functions that a CLIP system MUST implement for a given programming language.

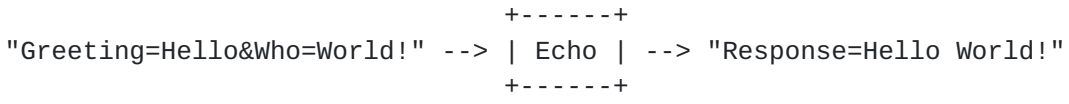
```
// CLIP in function
function clip_in(str,limit) {
  var in_array = new Array();
  var i = 0;
  var start = 0;
  var next = 0;

  if (typeof limit != "undefined") {
   for (i=0;i<limit;i++) {
    next = str.indexOf("&",start);
    in_array[i] = split_nv(str.substring(start,next));
    start = next + 1;
    }
   in_array[limit] = str.substring(start);
  } else {
    in_array = str.split('&');
    for (x in in_array) {
       in_array[x] = split_nv(in_array[x]);
    }
  }
 return in_array;
}

// CLIP out function
function clip_out(out_array) {

  for (x in out_array) {
    if (out_array[x].name && out_array[x].value) {
     out_array[x] = clip_encode(out_array[x].name) + "=" +
                    clip_encode(out_array[x].value);
    } else {
      if (out_array[x].name) {
       out_array[x] = clip_encode(out_array[x].name);
      } else {
       out_array[x] = clip_encode(out_array[x].value);
      }
    }
  }

  var output = out_array.join('&');
  return output;
}
```

```
   // Create a name / value pair object
   function nv_pair(n,v) {
    this.name = n;
    this.value = v;
   }

   // Split a string into a name / value pair object
   function split_nv(nv_str) {
     var nv = nv_str.split('=');
     if (nv.length == 2) {
       nvp = new nv_pair(clip_decode(nv[0]),clip_decode(nv[1]));
     } else {
       nvp = new nv_pair(null,clip_decode(nv_str));
     }
     return nvp;
   }

   // Decodes equals and ampersands
   function clip_decode( value ) {
     var x = trim ( value );
     x = x.replace(/%25/g, "%");
     x = x.replace(/%26/g, "&");
     x = x.replace(/%3D/g, "=");
     return x;
   }

   // Encodes equals and ampersands
   function clip_encode( value ) {
     var x = value;
     x = x.replace(/%/g, "%25");
     x = x.replace(/&/g, "%26");
     x = x.replace(/=/g, "%3D");
     return x;
   }
```

3.3 Example Message Definition

CLIP messages MAY be defined by providing a syntax, any sub-parameters and (optionally) return parameters. The example below demonstrates a simple request/response message set that corresponds to the input and output of a simple echo system.

Echo Example:



1. Greeting

SYNTAX
Greeting=<greeting_type>&<subparams>

GREETING TYPES

Valid Values	Description
"Hello"	Salutation

RETURNS

Name	Values
"Error"	"Invalid Input"

1.1 Hello

SYNTAX
Greeting=Hello&<subparams>

SUBPARAMS

Name	Valid Values
"Who"	<name_string>

RETURNS
Upon Success return:

Name	Values
"Response"	<response_string>

Upon error return:

Name	Values
Error	"Invalid Input"

3.4 Additional Notes and Recommended Practices

3.4.1 Case Sensitivity

CLIP names and values are case sensitive.

3.4.2 Duplicate Names

Only the first name in a set of duplicate names within a CLIP message SHOULD be processed. For example, if a CLIP message contains "price=1.00&price=5.00", the logical system parsing the name value pairs would only act upon "price=1.00" and discard "price=5.00".

3.4.1 Ordering Constraints

There are no specific ordering constraints on CLIP inputs or outputs unless specifically defined within a system's protocol definition. It is preferred that input and output sets be defined in a manner such that any such ordering constraints are avoided.

4. Security Considerations

There are no security considerations relevant to this document.

5. IANA Considerations

No actions are required from IANA as result of the publication of this document.

6. Acknowledgements

This document has benefited greatly from the comments of Terry Brogan, Pat Leary, Bill Franks and Dinkar Bhat.

7. References

7.1. Normative References

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, January 2008.

[RFC3875] Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, October 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

Authors' Addresses

Bob Booth
Motorola Mobility
101 Tournament Drive
Horsham, PA 19040
EMail: bob.booth@motorola.com