BLGW | Home Integration Protocol

Ikatu

2014-12-02

Contents

1 | Revision history:

Date	Author	Comments
[2014-05-15 Thu]	IK-JUP	First draft for review.
[2014-05-19 Mon]	IK-ALC	First review, added comments.
[2014-05-27 Tue]	IK-ALC	Added implementation details.
[2014-05-28 Wed]	IK-JUP	Added state query.
[2014-05-29 Thu]	IK-ALC	Removed comments, added more return codes.
[2014-07-29 Tue]	IK-FNA	Clean up.
[2014-12-02 Mar]	IK-SEG	Add location events command

2 | Introduction

With the introduction of BLGW, there is a standard way of identifying resources and specifying activity and state in the connected systems. Such activity can be represented unambiguously in the form of a text string.

Home Integration Protocol allows external applications to directly interact with BLGW. This is done by means of a minimalist line-based protocol that directly transports the text representation of all activity.

Users of HIP include mobile applications such as BeoLink App, home and building automation controllers, event monitoring and logging applications, and other special applications.

3 | Interface support and details

3.1 | RS232 connection

An RS232 interface connected to BLGW can be assigned for HIP functionality.

An application using HIP over RS232 will not require authentication and can use the protocol directly. Access to all zones is allowed.

Settings for the communication are 8 bits, no parity, 1 stop bit. Bit rate ranges from 9600 bps to 115200 bps.

3.2 | TCP/IP connection

HIP use a TCP socket and messages are packaged by lines. Each line is terminated with the sequence CR + LF.

Authentication is always required for TCP connections, and is specified in the following section.

Connection initiation is as follows:

1. Client establishes TCP connection to server.
2. Server initiates authentication loop.
    - On authentication success, HIP interchange begins.
    - On authentication error, the TCP connection is closed.

Note: Future version of this protocol will also provide a SSL connection using SSL authentication.

3.3 | Authentication

When a new TCP HIP connection is established, the server (BLGW) will start an authentication loop.

The procedure is as follows:

1. Server sends the string `login:␣` (ends with a space, ASCII 0x20).
2. Client sends the user name, followed by `CR+LF`.
3. Server sends the string `CR+LF` + `password:␣` (ends with a space, ASCII 0x20).
4. Client sends the corresponding password, followed by `CR+LF`. Server echo is disabled during password entry.
5. Server sends `CR+LF`.
6. If authentication succeeds, continue to HIP interchange.
7. On failure, server sends `error` + `CR+LF`, waits 2 seconds and repeats from setp 1 up to a total of 3 tries.

4 | Identification of resources, commands or events

A resource is uniquely identified by the combination of *area*, *zone*, *type* and *name*, and is represented uniquely in string form as a path with the form area/zone/type/name. For example:

```
Guest house/Kitchen/AV_RENDERER/BeoVision/
```

An event or command is represented by a resource path followed by an action (event or command), optionally followed by attributes and values.

Example of a simple command, and a command with 2 attributes:

```
Guest house/Kitchen/BUTTON/Lights ON/PRESS
Guest house/Kitchen/AV_RENDERER/BeoVision/Beo4 command?Command=TV&
       Destination selector=Video_source
```

Example state change event, with 1 attribute.

```
Guest house/Kitchen/BUTTON/Lights ON/STATE_UPDATE?STATE=1
```

Example generic event matching all state updates (see documentation for *generic programming*):

```
*/*/*/*/STATE_UPDATE
```

5 | HIP data interchange

The following commands are supported on HIP:

Command	Arguments	Description
`c`	Generic ID	Command, from client to server.
`f`	Generic resource	State filter request, client to server.
`e`	Code, message	Error code, server to client.
`q`	Generic resource	State query, client to server.
`s`	Specific ID	State update, server to client.
`r`	Specific ID	State response, client to server.
`l`	Location event ID	Could be `arrive` or `leave`.

Note: Event notifications other than user arrive and leave are not supported in this version of the protocol.

All commands from client to server take a single argument, which is an identifier for resources, commands or events.

A complete message consists of:

1. The command (1 character)
2. Space (ASCII 0x20)
3. The argument, which is an encoded string
4. Line termination, consisting of `CR+LF` (the server will also accept a single `CR`)

For example, to press all buttons in the installation, the client sends (do not try this at home):

```
"c */*/BUTTON/*/PRESS" + CR + LF
```

All commands from server to client may include further information, such as error codes.

5.1 | Command

The client application can send a command for BLGW to act on one or more resources. Generic specifications are allowed.

The following examples act on a specific dimmer, and on all products named *BeoVision*:

```
c Social/Dining/DIMMER/Chandelier/SET?LEVEL=60
c */*/AV_RENDERER/BeoVision/All standby
```

The server will answer with an *e* as soon as it processes the command.

5.2 | State query

The state of a resource or group of resources can be queried at any time. Normally, the client application will query the state of some resources in order to render a user interface with representations for those resources.

Resource state responses will be returned as a block, with no further state updates in between. The block of responses correspond to a snapshot of the resources at the time the query was processed by the server.

Example query for all resources in a zone:

```
q Social/Dining/*/*
```

The server will answer with an *e* as soon as it processes the command, followed by the corresponding responses if the command succeeds:

```
r Social/Dining/DIMMER/Downlight/STATE_UPDATE?LEVEL=33
r Social/Dining/AV_RENDERER/BeoSound/STATE_UPDATE?...
r Social/Dining/BUTTON/Lights on/STATE_UPDATE?STATE=1
r Social/Dining/BUTTON/Party mode/STATE_UPDATE?STATE=0
```

5.3 | State filter request

In order to receive state update notifications, the client must specify a filter. The filter consists of a generic path to a group of resources.

The following example will match all state updates from the zones Kitchen or Dining:

```
f */Kitchen|Dining/*/*
```

The following example will match all dimmer state updates:

```
f */*/DIMMER/*
```

To disable state update notifications, send the command with no arguments:

```
f
```

By default, HIP connections start with state notifications disabled.

The server will answer with an *e* as soon as it processes the command.

5.4 | State update notification

Whenever the state of a resource changes, the server will generate a state update.

The client will only receive state update notifications for resources that match the state filter set and are accessible by the authenticated user.

Example state update:

```
s Social/Dining/BUTTON/Party mode/STATE_UPDATE?STATE=3
```

5.5 | Location events

The client application can send location events to notify BLGW the user arrived or left the home.

The following examples show the two possible location events:

```
"l leave" + CR + LF
"l arrive" + CR + LF
```

The first one means the user left, and the second one means the user arrived.

The server will answer with an *e* as soon as it processes the command.

5.6 | Return code messages

A message sent to the server may result in an error. The server will generate an error response which consists of an error code followed by the received message.

The only exception is for the *message too long* error, which will not include the message in the reply.

In any case, if an error condition arises, the originating message will not be processed.

Error codes:

Code	Meaning
OK	Command successfully processed.
CMD	Wrong / unrecognized command.
SYN	Bad syntax, or wrong character encoding.
ACC	Zone access violation.
LEN	Received message too long.

Example responses from server:

```
e OK c global/global/BUTTON/*/PRESS
e ACC c */*/BUTTON/*/PRESS
```

6 | Access restrictions

Each user has a list of zones where he can interact. Zones not in this list cannot be accessed via HIP for that user.

Therefore, unless the user has access to all zones that are included in the message, generic commands are not allowed and will generate an access violation error.

The following list shows some commands and the condition needed for them to succeed:

- `c */*/BUTTON/*/PRESS`: The user must have the "all zone access" flag set.
- `c global/*/BUTTON/*/PRESS`: The user must have the "all zone access" flag set.
- `c */global/BUTTON/*/PRESS`: The user must have the "all zone access" flag set.
- `c global/bedroom|kitchen/BUTTON/*/PRESS`: The user must access to global/bedroom and global/kitchen.
- `c global/kitchen/BUTTON/*/PRESS`: The user must access to global/kitchen.

7 | Encoding of messages

All zone, resource, command and event identifiers have the format of a relative URI (uniform resource identifier):

- Identifiers have the form of a URI path, with a hierarchic list of symbols joined by forward slashes.
- Arguments to commands and events also comply with URI queries, with a list of attributes and corresponding values.

When it comes handling special characters, such as the space (ASCII 0x20) or multilingual characters, it is necessary to avoid ambiguity by encoding the path as a single string (no spaces), and all in plain printable ASCII characters.

The same applies to *reserved characters*, such as / or ?, which have special meaning as delimiters in the URI, and must be encoded if they appear elsewhere.

This specification adopts the same encoding as specified in section 2 of IETF RFC 3986, commonly known as *URL encoding* or *percent encoding*.

Rules are basically as follows:

- *Percent encoding* of a byte is the percent sign % followed by the two upper-case hexadecimal digits of the byte. For example, the percent sign (ASCII 0x25) *must* be encoded as %25. Similarly, the byte 0xE3 is encoded as %E3.
- Special characters must be percent encoded.
- Multilingual characters must be represented in UTF-8, and each of the corresponding bytes should then be percent encoded if needed.

In the direction server to client, the encoding is always done according to this specification (uppercase hexadecimal, only percent encode special characters).

The client, however, has more flexibility when sending messages to the server:

- The server will accept both upper or lower case hexadecimal digits.
- The server will accept UTF-8 characters directly. Note that Telnet encoding must be taken care of anyway for a TCP connection.
- Any character (not only special characters) can be percent encoded.

For example, the following messages are equivalent:

```
c Up/Salón/BUTTON//PRESS
c Up/Salón/BUTTON//PRES%53
c Up/Sal%C3%B3n/BUTTON/%d1%81%d0%b2%d0%b5%d1%82/PRESS
c Up/Sal%C3%B3n/BUTTON/%D1%81%D0%B2%D0%B5%D1%82/PRESS
```

8 | Glossary and references

BLGW BeoLink Automation Gateway.

HIP Home Integration Protocol

Server BLGW or, in general, the product / application accepting HIP connections from clients, processing commands and returning events and state.

Client Application or product establishing a HIP connection to BLGW (server).

Telnet Protocol for terminal communication over Internet Protocol defined in RFC 854: `https://tools.ietf.org/html/rfc854`

Echo Repetition of characters received on a connection back to the sender. For telnet, this is pecified in RFC 857: `https://tools.ietf.org/html/rfc857`

CR Carriage return ASCII 0x0D.

LF Line Feed ASCII 0x0A.

Line termination The sequence CR + LF. The server also accepts a single incoming CR as line termination.

URL encoding ASCII encoding of URIs as specified in RFC 3986: `http://tools.ietf.org/html/rfc3986`