UDAP Specifications (For Second Screen TV and Companion Apps)

This document describes the UDAP (Universal Discovery & Access Protocol) concept and provides the protocol specification and service profiles in detail. Developers can acquire the knowledge needed to implement the Controller application to control the LG Smart TV.

LG UDAP 2.0 Protocol Specifications

This section describes the UDAP concept and the protocol specification in detail. In this document, developers can acquire the protocol-related knowledge needed to implement the Controller application to control the LG Smart TV.

LG UDAP 2.0 Service Profiles

This section describes the actual operation of the service utilizing the UDAP and the details of the API.

Note

UDAP 2.0 is only supported in NetCast 3.0 (LG Smart TV models released in 2012) and NetCast 4.0 (LG Smart TV models released in 2013) platforms.

This document is also provided in Korean. You can download it from [Documentation > Technical Notes].

Contents

LG UDAP 2.0 Protocol Specifications

This document describes the Universal Discovery & Access Protocol (UDAP) concept and the protocol specification in detail. In this document, developers can acquire the protocol-related knowledge needed to implement the Controller application to control the LG Smart TV.

UDAP Overview

UDAP (Universal Discovery & Access Protocol) is a HTTP/1.1-based protocol defined by LG, that is intended to provide the environment to enable users to control fast-evolving smart home appliances, including smart TV, by using various devices such as a smart phone or tablet PC, and share their contents.

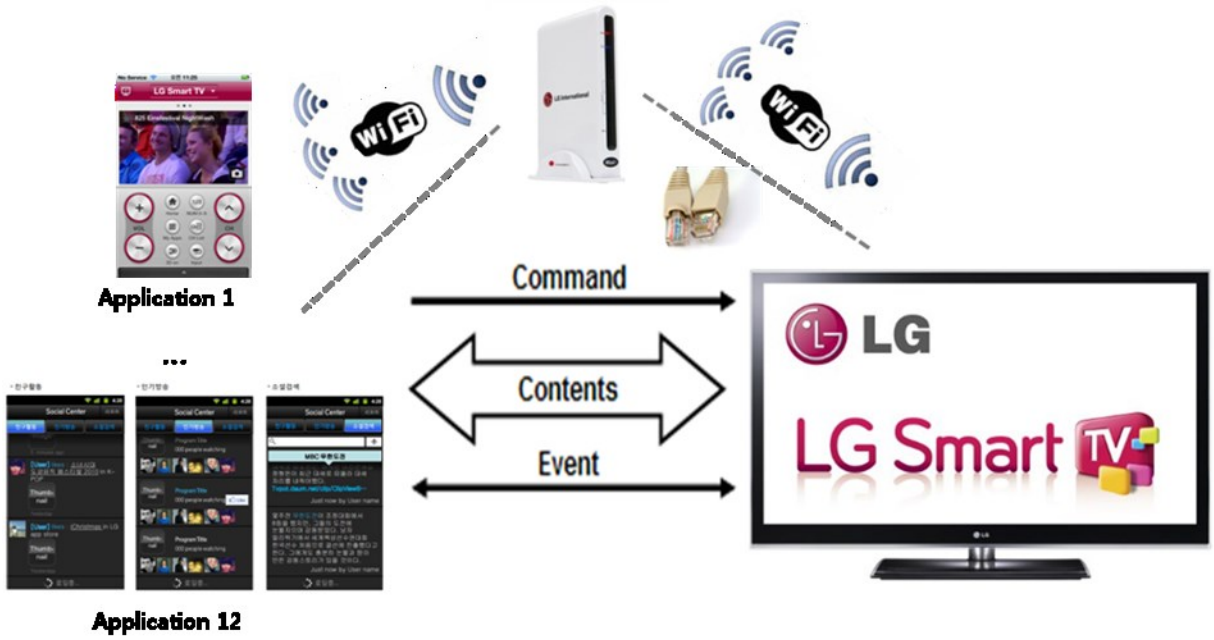
UDAP allows smart device application developers to develop various applications as they want by utilizing LG products, and allows users to utilize LG products more easily and in various ways by downloading the applications suitable for their smart devices.

The figure below displays the communication flow between a smart TV and a smart phone in diagram form.

• Controller: A device that provides users with a certain service application with user interfaces
• Host: A device that communicates with a Controller and provides a service requested by the Controller

A Controller and a Host must both be connected to the same local network through a router for communicating to each other, and up to 12 Controllers can connect to a Host simultaneously.

An LG smart TV acts as a Host and users can connect the LG smart TV to a router using wired technology or wirelessly.

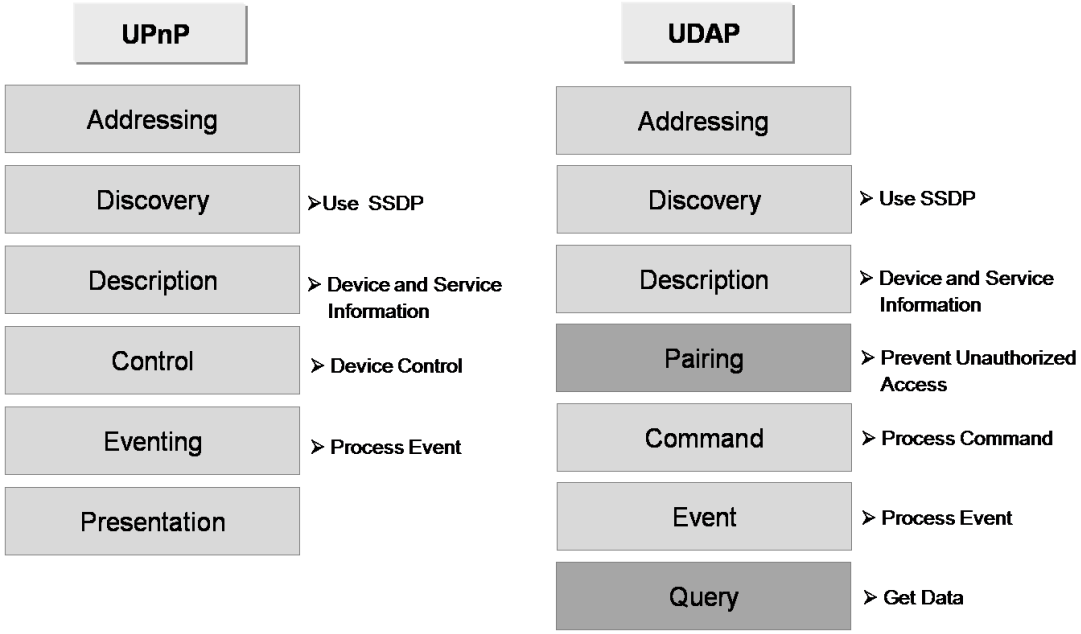


[Figure] Communication flow between devices

This chapter includes the following section.
• Protocol Stack

Protocol Stack

The UDAP uses the following protocol stack based on the communication concept of UPnP Device Architecture. As a Host, the LG smart TV is not involved in addressing because it delegates this to its network configuration software which is separately operated. A Controller such as a smart phone or tablet PC is also not required for addressing. The network configuration function of each device can be used for addressing.



[Figure] Conceptual diagram of protocol stack - UPnP and UDAP

Note

UDAP is based on the UPnP architecture concept, but it is different from UPnP.

The conceptual diagram above consists of the following protocol stacks from the perspective of software configuration.

<table>
<tr><td rowspan="3">SSDP M-SEARCH</td><td>Pairing</td><td>Command</td><td>Event</td><td>Description</td><td>Query</td></tr>
<tr><td colspan="3">HTTP POST</td><td colspan="2">HTTP GET</td></tr>
<tr><td colspan="5">HTTP/1.1</td></tr>
<tr><td>UDP</td><td colspan="5">TCP</td></tr>
<tr><td colspan="6">IP</td></tr>
</table>

[Figure] Protocol stack

Based on the protocol stack in the figure above, each of Discovery/Description/Pairing/Command, Event and Query has a role and significance as follows:

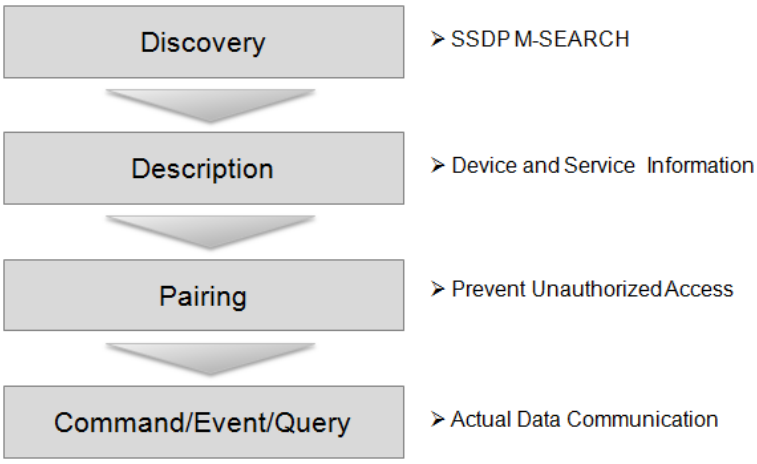
[Table] Protocol stack type

Protocol stack type	Description
Discovery	It is used to search for a device. A Controller sends the SSDP request and a Host sends the SSDP response.
Description	It is used to get the description of the searched service. The M-SEARCH response header of Discovery informs a Controller of the URL where the description can be found, in the Location field.

Protocol stack type	Description
Pairing	This confirms that a Controller is the valid device which can access a Host.
Command	A Controller sends a command to control a Host.
Event	A Controller and a Host send an event to each other when the status of a counterpart is changed or a specific event occurs.
Query	A Controller uses a query to obtain the data service information provided by a Host.

The Description/Pairing/Command, Event or Query described in the table above uses the XML-based request and response; the character encoding of all the XML documents uses UTF-8.

The figure below displays the communication process between a Controller and a Host in diagram form.



[Figure] Communication process

Discovery

This chapter describes how a Controller searches for a Host using SSDP M-SEARCH.

Although the UDAP uses the same method as the SSDP M-SEARCH that is used in the UPnP, the value specified in the ST (Search Target) of M-SEARCH is slightly different from the method defined in the UPnP.

The following protocol stacks are used to search for a Host using SSDP.

<table>
<tr><td>SSDP M-SEARCH
(Standard SSDP)</td><td>SSDP B-SEARCH
(Broadcast-based SSDP)</td></tr>
<tr><td>Multicast</td><td>Broadcast</td></tr>
<tr><td colspan="2">UDP</td></tr>
<tr><td colspan="2">IP</td></tr>
</table>

[Figure] Search protocol stack

This chapter includes the following sections.

M-SEARCH Request

A Controller sends the M-SEARCH method of SSDP in multicast to search for a Host that is connected to the same local network. The multicast address used is **239.255.255.250** and the port is **1900**. These values are the same as those defined in UPnP.

A Controller uses the following format defined in SSDP when sending M-SEARCH. A value in Italic font can actually be specified by a Controller.

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: Maximum time (in seconds) to wait for response of host
ST: URN value of service to search
USER-AGENT: OS/version UDAP/2.0 product/version
```

The M-SEARCH request starts with M-SEARCH * HTTP/1.1 and all the lines of a header end with “\r\n”. The last line ends with “\r\n”. For example, the actual M-SEARCH request of a Controller is sent in the following format:

```
M-SEARCH * HTTP/1.1\r\n
HOST: 239.255.255.250:1900\r\n
MAN: "ssdp:discover"\r\n
MX: 3\r\n
ST: udap:rootservice\r\n
USER-AGENT: iOS/5.0 UDAP/2.0 iPhone/4\r\n
\r\n
```

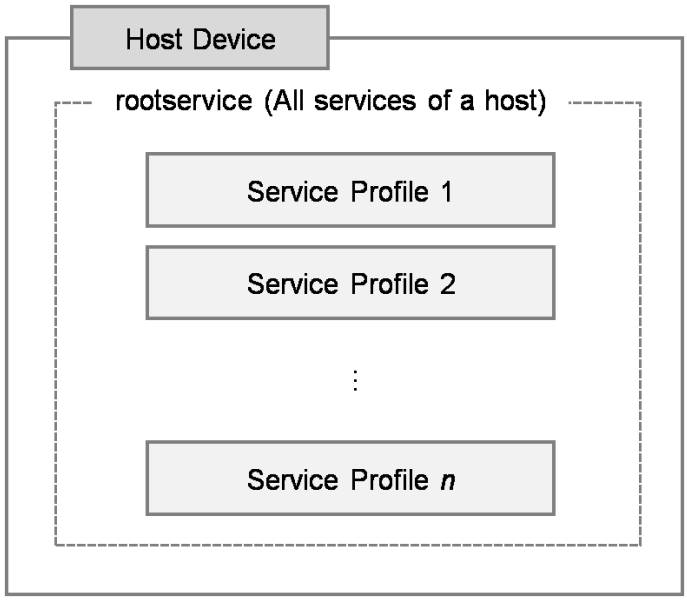
Each header field in the M-SEARCH request means the following:

[Table] M-SEARCH request header

Header	Description

Header	Description
HOST	**Must be included** The field value consists of a multicast address and a port that are used in the SSDP and must be the same as the following: 239.255.255.250:1900
MAN	**Must be included** The field value specifies the value of "ssdp:discover" to be used for the search in SSDP.
MX	**Must be included** Maximum time (seconds) to wait for the M-SEARCH response. It must be greater than 1, but less than 5 seconds according to the UPnP standard. Considering that several Hosts can be connected to the network, **3 seconds or longer is recommended.**
ST	**Must be included** This specifies the Search Target to search for using M-SEARCH. The Search Target is described below.
USER-AGENT	**Must be included** An optional header in UPnP, but a required header in UDAP. This uses the OS/version UDAP/2.0 product/version format. The OS and product information can be omitted, but **UDAP/2.0 must be included.** USER-AGENT USER-AGENT: iOS/5.0 UDAP/2.0 iPhone/4 USER-AGENT Android/4.0 UDAP/2.0 USER-AGENT UDAP/2.0

A ST value that specifies a target to search for using M-SEARCH is determined by the service configuration of a target Host. A Host supporting UDAP usually supports more than one service. Service Profile is an application comprising each of these services. The figure below shows the concept of the service profile configuration of a Host.



[Figure] Service profile configuration of a Host

In the figure above, rootservice means all the services in a Host and each service profile has a different service name. A Controller can search for a Host by specifying a URN that corresponds to the name of a rootservice or each service profile.

<ST value>

- udap:rootservice
  Searches all the services of a Host and all the devices that support UDAP regardless of the service profile type.

- urn:schemas-udap:service:serviceName:version
  Searches for a Host that has a service profile corresponding to serviceName.
  For the details of the supported service profiles, refer to **LG UDAP 2.0 Service Profiles**.

Note

If the information of several service profiles is accessed for creating a Controller application program, use udap:rootservice to search for a Host.

For example, the actual M-SEARCH request of a Controller that searches for a Host that supports a service profile where the text input is supported is as follows:

```
M-SEARCH * HTTP/1.1\r\n
HOST: 239.255.255.250:1900\r\n
MAN: "ssdp:discover"\r\n
MX: 3\r\n
ST: urn:schemas-udap:service:smartText:1\r\n
USER-AGENT: iOS/5.0 UDAP/2.0 iPhone/4\r\n
\r\n
```

M-SEARCH Response

A Host that received the M-SEARCH request analyzes the ST value and then if the M-SEARCH request of a Controller is valid, it sends the following response to the Controller using the UDP Unicast transport mechanism using the source IP address and the port of the Controller. If the ST value of M-SEARCH is not valid, the Host does not send the response to the Controller. The M-SEARCH response starts with HTTP/1.1 200 OK and all the lines of a header end with “\r\n”. The last line ends with “\r\n”. A Controller can get an IP address that can be used to communicate with a Host by getting the source IP of the M-SEARCH response.

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = available time (in seconds) that Controller can communication
with Host
DATE: Time when the response is occurred
EXT:
LOCATION: HTTP that service can get description
SERVER: OS/version UDAP/2.0 product/version
ST: ST value that Controller requested
USN: composite identifier of M-SEARCH response
```

An example of the actual response of a Host, described in M-SEARCH Request, which has a service profile that supports text input is as follows:

```
HTTP/1.1 200 OK\r\n
CACHE-CONTROL: max-age=172800\r\n
DATE: Wed Jul 11 05:55:53 2012 GMT\r\n
EXT: \r\n
LOCATION: http://192.168.10.51:8080/udap/api/data?target=smartText.xml\r\n
SERVER: Linux/2.6.18-308.4.1.el5 UDAP/2.0 LGSmartTV_33/2.0\r\n
ST: urn:schemas-udap:service:smartText:1\r\n
USN: uuid:33068e81-3306-0633-619b-9b61818e0633::urn:schemas-udap:service:smartText:1\r\n
\r\n
```

Each header field in the M-SEARCH response means the following:

[Table] M-SEARCH request header

Header	Description
CACHE-CONTROL	Valid time (in seconds) for communication with a Host. UDAP uses very high values in this header. (172800 sec.) A value is determined and forwarded using a directive called "max-age".
DATE	Time in GMT when the response of a Host is created
EXT	A header that is set up in UPnP/1.1 for UPnP 1.0 compatibility. It is also added to a response according to UPnP convention in UDAP. There is only a header name, but no header value.
LOCATION	It specifies a URL value that can be used to get the description of a searched service using M-SEARCH and this header value is in the absolute URL format.

Header	Description
SERVER	Complies with the format of OS/version UDAP/2.0 product/version. It has the server information of a Host that supports UDAP.
ST	This is the ST (Search Target) value that is sent by the M-SEARCH request and its format is as follows: • udap:rootservice • urn:schemas-udap:service:serviceName:version
USN	A header that has a Unique Service Name. The value of a header is created in the following format using the values of UUID (Universal Unique IDentifier) and ST (Search Target). • uuid:uuid_of_Host_device::ST_of_response

After a Host sends a response to a Controller for a valid ST, it shows the following discovery results on its display to help a Controller detect the Host.

[Figure] Host screen after discovery response

Extensions (B-SEARCH)

The B-SEARCH is an SSDP-based extension that supports the discovery of a Host that supports UDAP when it is connected to a router which does not support multicast. When a request is transmitted, **the broadcast address of a network** (255.255.255.255 or the broadcast address of a network interface) is used instead of a multicast address (239.255.255.250), and **port 1990** is used instead of port 1900.

```
% /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:4E:9E:64
          inet addr:192.168.10.51  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4e:9e64/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:175 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20265 (19.7 KiB)  TX bytes:21976 (21.4 KiB)
```

In addition, B-SEARCH (Broadcast-SEARCH) is used instead of M-SEARCH for a method name and 255.255.255.255:1990 must be used as the value of a Host. The mechanism of request and response is exactly the same as the description in M-SEARCH Request and M-SEARCH Response.

Note

M-SEARCH may not work on some routers while B-SEARCH will work on all routers. Therefore, if there is no Host response for the MX value of M-SEARCH that is sent by a Controller, the Controller re-sends B-SEARCH to search for a Host again.

The format of the B-SEARCH request is as follows: Note that the method name is changed to B-SEARCH and the value of a Host is changed to broadcast_address:1990.

```
B-SEARCH * HTTP/1.1
HOST: 255.255.255.255:1990
MAN: "ssdp:discover"
MX: Maximum time to wait host response (in seconds)
ST: URN value of service to search
USER-AGENT: OS/version UDAP/2.0 product/version
```

The example of the actual B-SEARCH request and response is similar to those of M-SEARCH as below, and the response from a Host is the same as that described in in M-SEARCH Response.

[B-SEARCH Request]

```
B-SEARCH * HTTP/1.1\r\n
HOST: 255.255.255.255:1990\r\n
MAN: "ssdp:discover"\r\n
MX: 3\r\n
ST: urn:schemas-udap:service:smartText:1\r\n
USER-AGENT: iOS/5.0 UDAP/2.0 iPhone/4\r\n
\r\n
```

[B-SEARCH Response]

```
HTTP/1.1 200 OK\r\n
CACHE-CONTROL: max-age=172800\r\n
DATE: Wed Jul 11 05:55:53 2012 GMT\r\n
EXT: \r\n
LOCATION: http://192.168.10.51:8080/udap/api/data?target=smartText.xml\r\n
SERVER: Linux/2.6.18-308.4.1.el5 UDAP/2.0 LGSmartTV_33/2.0\r\n
ST: urn:schemas-udap:service:smartText:1\r\n
USN: uuid:33068e81-3306-0633-619b-9b61818e0633::urn:schemas-udap:service:smartText:1\r\n
\r\n
```

Implementation Example

The following example code is an example of a request or a response reception of a simple SSDP M-SEARCH in Linux. This example is created on the assumption that there is only one Host in a local network. However, there may be several Hosts in a network. In addition, the codes for exception handling and the parsing of the response header must be created.

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    int s;

    if ((s = socket(AF_INET, SOCK_DGRAM, 0)) > 0) {

        char buf[] = "M-SEARCH * HTTP/1.1\r\n"\
                     "HOST: 239.255.255.250:1900\r\n"\
                     "MAN: \"ssdp:discover\"\r\n"\
                     "ST: udap:rootservice\r\n"\
                     "MX: 3\r\n"\
                     "USER-AGENT: Linux/2.6.18 UDAP/2.0 CentOS/5.8\r\n\r\n";
        struct sockaddr_in srv;

        bzero(&srv, sizeof(srv));
        srv.sin_family = AF_INET;
        srv.sin_port = htons(1900);
        srv.sin_addr.s_addr = inet_addr("239.255.255.250");

        if (sendto(s, buf, strlen(buf), 0, (struct sockaddr *) &srv, sizeof(srv)) > 0) {

            fd_set readset;
            struct timeval tv;

            printf("%s\n", buf);

            tv.tv_sec = 3;
            tv.tv_usec = 0;
            FD_ZERO(&readset);
            FD_SET(s, &readset);

            if (select(s + 1, &readset, NULL, NULL, &tv) > 0) {

                char res[512+1];
                struct sockaddr_in from;
                socklen_t fromlen = sizeof(struct sockaddr_in);
                int recvLen = -1;

                if ((recvLen = recvfrom(s, res, sizeof(res)-1, 0, (struct sockaddr *) &from, &fromlen)) > 0) {

                    res[recvLen] = '\0';
                    printf("got response from: (%s)\n", inet_ntoa(from.sin_addr));
                    printf("%s", res);
                }
            }
        }

        close(s);
    }

    return 0;
}
```

The result of the above example code is as follows:

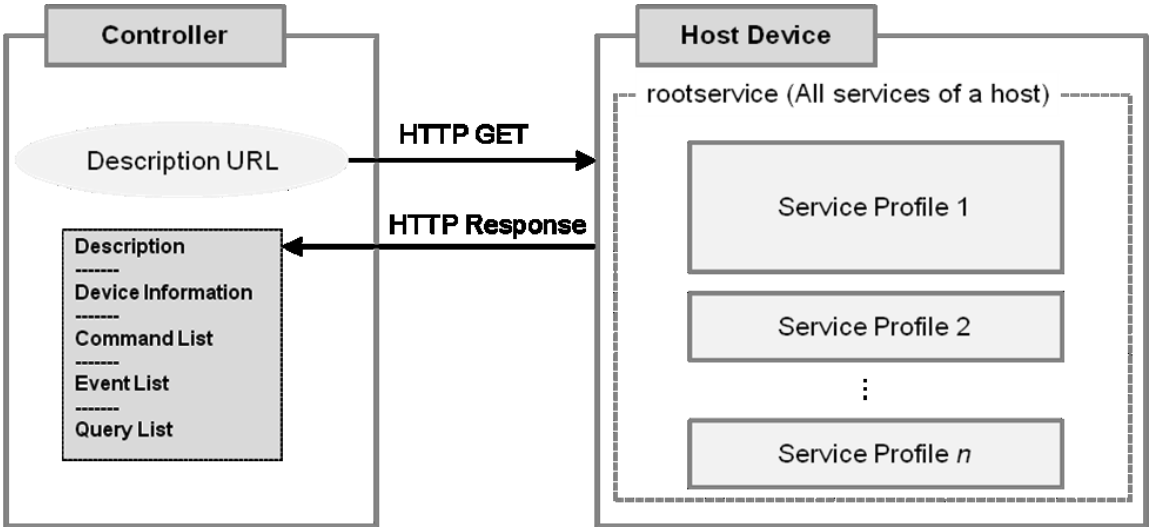
```
% ./msearch
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
ST: udap:rootservice
MX: 3
USER-AGENT: Linux/2.6.18 UDAP/2.0 CentOS/5.8


got response from: (192.168.10.51)
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=172800
DATE: Sat Jul 14 07:08:34 2012 GMT
EXT:
LOCATION: http://192.168.10.51:8080/udap/api/data?target=rootservice.xml
SERVER: Linux/2.6.18-308.4.1.el5 UDAP/2.0 LGSmartTV_33/2.0
ST: udap:rootservice
USN: uuid:33068e81-3306-0633-619b-9b61818e0633::udap:rootservice

ssul97@ssul(tty1) [Linux 2.6.18] ~
```

Description

This chapter describes the information about a device that supports a service searched in the Discovery and the Description which is used to obtain the list of Command/Event/Query.

The Description is described in XML format and is obtained by requesting the absolute URL in the LOCATION field of the SSDP M-SEARCH response header to a Host using HTTP GET.

The conceptual diagram showing the process for obtaining the Description for Service Profile 1 is shown below:

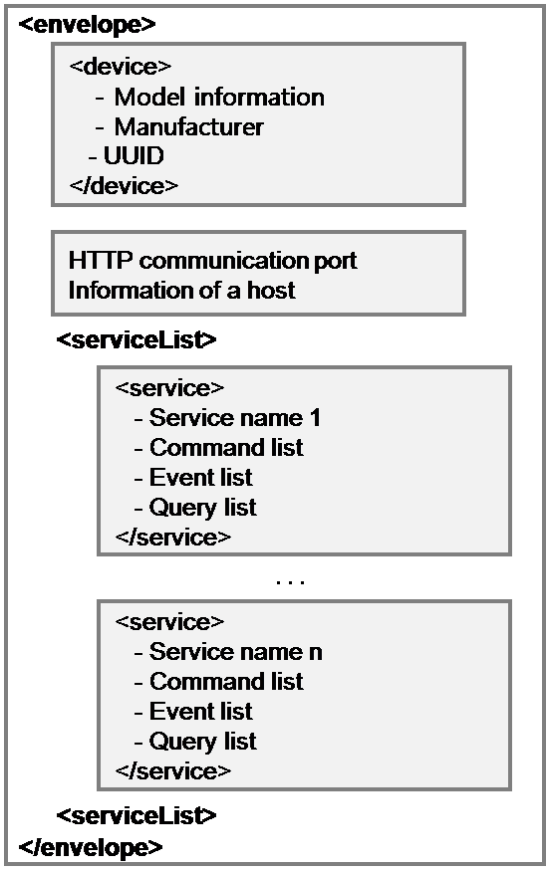


[Figure] Conceptual diagram of description process

This chapter includes the following sections.
- Structure of Description Document
- Examples of Description Document
- Utilization of Description Document

Structure of Description Document

The description of UDAP is defined in the XML document and the configuration diagram of the document is shown below.



[Figure] Structure of description document

The Description document provides the function of a corresponding service according to the ST value of the M-SEARCH request.

- **udap:rootservice**
  The information of command/event/query of all the services supported by a Host is described and several services are described in the serviceList.

- **urn:schemas-udap:service:serviceName:version**
  Only the information for command/event/query of a specific service is described and only one service is described in the serviceList.

The document structure of the Description is displayed in an actual XML document as below. The bold black font is a fixed value and the red Italic font may vary depending on Host.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <device>
      <deviceType>type of Host device</deviceType>
      <modelName>model name of Host device</modelName>
      <friendlyName>user-friendly name of a model</friendlyName>
      <manufacturer>LG Electronics</manufacturer>
      <uuid>Host device UUID value</uuid>
   </device>
   <port> HTTP communication port of a Host device</port>
   <serviceList>
      <service name="service name 1">
         <apiList>
            <api type="command">
               <name>command name 1</name>
               <name>command name 2</name>
               <!-- additional commands are listed if available -->
            </api>
            <api type="event">
               <name>event name 1</name>
               <name> event name 2</name>
               <name> event name 3</name>
               <!— additional events are listed if available -->
            </api>
            <api type="query">
               <name>query name 1</name>
               <!-- additional queries are listed if available -->
            </api>
         </apiList>
      </service>
      <service name="service name 2">
         <!-- command / event / query list for service 2 -->
      </service>
         <!-- additional services are listed if available -->
   </serviceList>
</envelope>
```

As shown in the XML document, the apiList describes the API name of a Host per command/event/query and the information for each API call format is not described separately. **For more information about an actual API call format, refer to** LG UDAP 2.0 Service Profiles. The services described in LG UDAP 2.0 Service Profiles comply with the protocol standards in this document and only the call format and operations for the command/event/query API of each service are described.

Examples of Description Document

The XML document below shows an example of the description which can be obtained by search results of remote control key input and remote control service for a service in a Host.

For the description obtained as a search result by using udap:rootservice, the name and the apiList of an additional service can be added in the example below. In other words, the description obtained as the search result by using udap:rootservice describes the whole service list described in **LG UDAP 2.0 Service Profiles**. The related example is provided on the next page.

<xml version="1.0" encoding="utf-8">

```
▼<envelope>
  ▼<device>
      <modelName>LGSmartTV_33</modelName>
      <friendlyName>LG Smart TV</friendlyName>
      <manufacturer>LG Electronics</manufacturer>
      <uuid>33068e81-3306-0633-619b-9b61818e0633</uuid>
    </device>
    <port>8080</port>
  ▼<serviceList>
    ▼<service name="mobilehome">
      ▼<apiList>
        ▼<api type="command">
            <name>LGSmartWorld</name>
            <name>AppTerminate</name>
            <name>AppExecute</name>
          </api>
        ▼<api type="event">
            <name>LGSmartWorld_Event</name>
            <name>Mobilehome_App_Errstate</name>
            <name>Mobilehome_App_Change</name>
          </api>
        ▼<api type="query">
            <name>navigationContentDetail</name>
            <name>navigationContent</name>
            <name>navigationCategory</name>
            <name>navigationOrder</name>
            <name>appicon_get</name>
            <name>appnum_get</name>
            <name>applist_get</name>
          </api>
        </apiList>
      </service>
    </serviceList>
</envelope>
```

The example of an actual XML document of the description obtained as the search result using udap:rootservice is as follows.

```
<envelope>
  <device>...</device>
  <port>8080</port>
  <serviceList>
    <service name="AppToApp">
      <apiList>
        <api type="command">
          <name>SendMessage</name>
          <name>TerminateApplication</name>
          <name>LaunchApplication</name>
        </api>
        <api type="event">
          <name>ReceiveMessage</name>
        </api>
        <api type="query">
          <name>GetApplicationStatus</name>
          <name>GetApplicationID</name>
        </api>
      </apiList>
    </service>
    <service name="smartText">
      <apiList>
        <api type="event">
          <name>TextEdited</name>
          <name>KeyboardVisible</name>
        </api>
      </apiList>
    </service>
    <service name="mobilehome">
      <apiList>
        <api type="command">...</api>
        <api type="event">...</api>
        <api type="query">...</api>
      </apiList>
    </service>
    <service name="netrcu">...</service>
  </serviceList>
</envelope>
```

The following is the description of elements and attributes comprising the XML document in the above example.

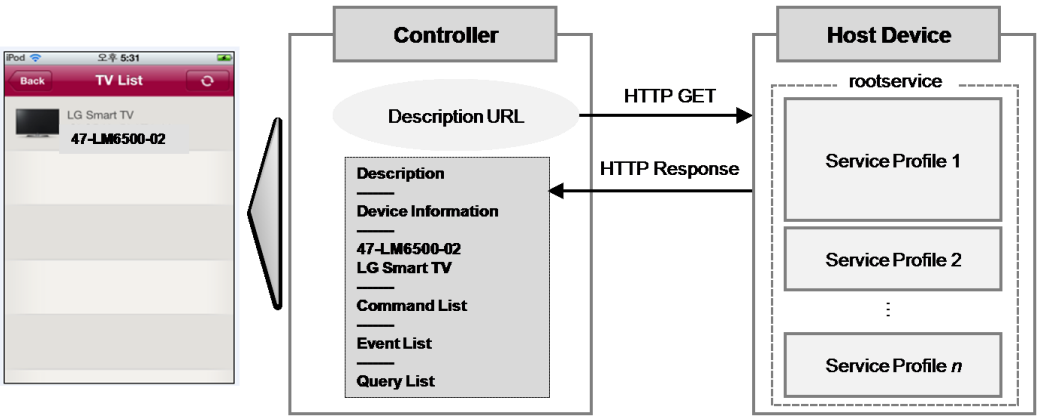
[Table] Elements of description document

XML element	XML attribute	Description
envelope	-	Start part of the description XML document
device	-	Descriptions of the information on a Host device
deviceType	-	Device type of the Host device. For LG Smart TV, the value is represented as “TV”.
modelName	-	Model name of a Host. Represented as "inch-model name_suffix". For example, 47-LM6500_32.
friendlyName	-	User-friendly name of a modelName. For LG Smart TV, the value follows the TV setting value. If no value is set in TV, it is represented as default value, “LG Smart TV”.
manufacturer	-	Manufacturer of a Host device. Fixed as LG Electronics.
uuid	-	Stands for Universal Unique IDentifier of a Host device. Each Host device in the same local network has a different unique value.
port	-	Communication port information regarding a Host device. A Controller sends a command/event/query to a Host using this port.
serviceList	-	Start part of the description for the service list of a Host device

XML element	XML attribute	Description
service	name	Start part of the description for one of the services of a Host. The service name is described in the name attribute.
apiList	-	Start part of the description for the command/event/query list that is supported by one of the services of a Host
api	type	Start part of the description for a command, event or query list that is supported by one of the services of a Host. The value of the type attribute is one of command, event or query.
name	-	Describes the actual name of the supported API for command, event or query that is supported by one of the services of a Host. For more information about the actual call format of each API, refer to LG UDAP 2.0 Service Profiles.

Utilization of Description Document

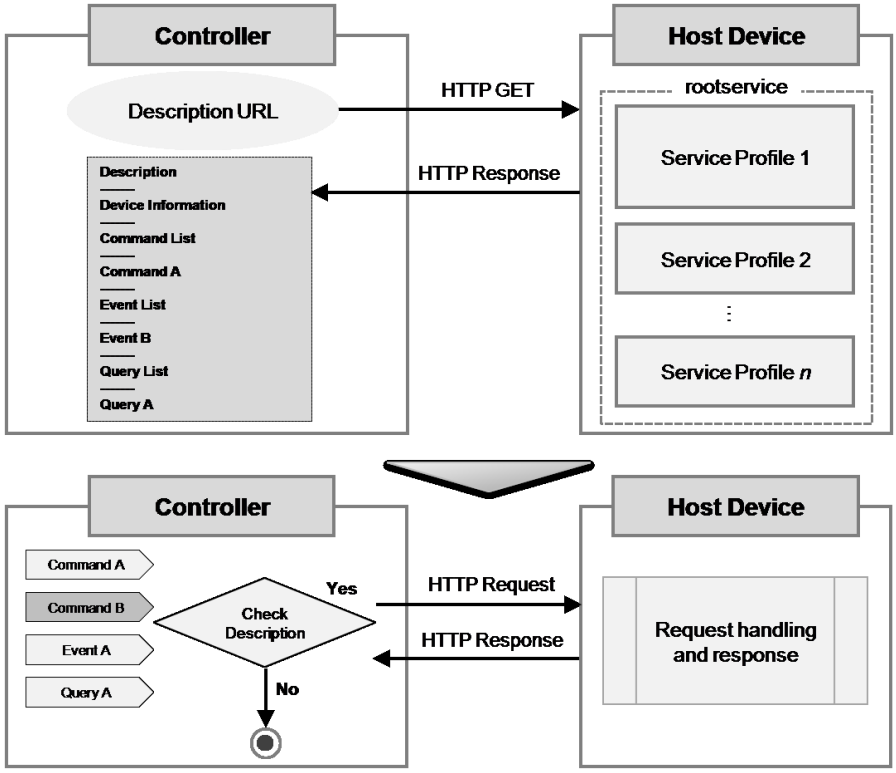
A Controller uses the description document to get the information that can be used to access a Host after discovery as well as the information that can be displayed on a user's screen. The figure below is an example of displaying the Discovery result on the user's screen of a Controller using the model name of Description.



[Figure] Display of the discovery screen of a Controller

Generally, even for Hosts that support the same service, the supported API list may be different depending on the detail model of a Host. In this case, the description document provides the list of the API name of command, event, or query that is supported by the service to guarantee the operational compatibility between a Controller and a Host. **The Controller must check the API list provided in the description and also check whether the function that the Controller wants to call is supported by the Host.** After that, by calling an API that is actually supported by the Host, prevent any malfunction or error. When a Host receives a call request from a Controller for an API that is not supported by the Host, it sends back an error response to the Controller.

The figure below displays the process that a Controller uses to check the API list in the description and communicate with a Host in diagram form. In addition, **a Controller must complete the pairing process described in the next chapter to call a Host API**.



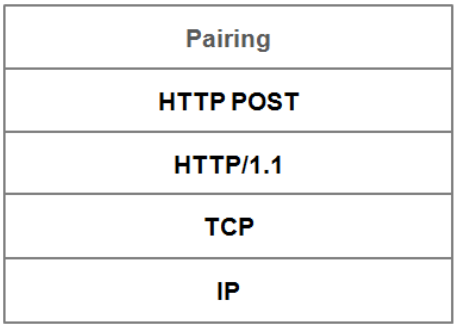
[Figure] Diagram of checking API support by a Controller

Pairing

This chapter describes the pairing process. Pairing is a kind of authorization process that allows only authorized Controllers to access the Host.

A Controller can call the command/event/query API of a Host only when it receives a success response from the Host by entering a pairing key displayed on the screen of a Host to access, and sending it to the Host.

Pairing uses the **IP address and port number that are obtained from Discovery and Description to send the pairing key using HTTP/1.1 POST**, and the following protocol stacks are used.



[Figure] Pairing protocol stack

Pairing provides the APIs described in the following sections. The path used for POST is /udap/api/pairing and the text/xml; charset=utf-8 is used for Content-Type. In addition, **a Controller must include UDAP/2.0 in the User-Agent header of POST**.
The XML format used for pairing is as follows. The black font is a fixed value and the red Italic font can vary per API.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="pairing">
        <name>apiName</name>
        <paramName>param value</paramName>
        <!-- additional parameters are listed if available -->
    </api>
</envelope>
```

Note

When building XML, avoid inserting the line change character (“\r”, “\n”) if possible. This reduces data transmission size by excluding unnecessary characters and speeds up the operation at the XML parser of a Host.

This chapter includes the following sections.

Request Pairing Key (Controller >> Host)

A Controller uses the following XML format when it requests a pairing key to a Host as HTTP POST.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="pairing">
        <name>showKey</name>
    </api>
</envelope>
```

A Host that receives a showKey request displays a 6-digit pairing key on its screen as shown below. The 6-digit pairing key is displayed on the screen with a space after every 2 digits for better readability. When a Controller requests the pairing, it must send the 6-digit string with no spaces. For example, in the figure below, the pairing key value that is forwarded by its actual pairing request is 513296.

[Figure] Host screen after pairing key request

The showKey checks any request errors and sends a response code to the Controller as shown in the table below:

[Table] Response code of showKey request

Response Code	Description

Response Code	Description
200	HTTP/1.1 200 OK The showKey request is successful.
400	HTTP/1.1 400 Bad Request The showKey request is transmitted in an incorrect format.
500	HTTP/1.1 500 Internal Server Error During showKey request handling, an internal handling error occurs in a Host.

The actual request of a Controller and the response of a Host using showKey are as follows. Although the actual request XML of a Controller does not have the new line character, it is included for better readability of the XML transmitted from the request in this document.

```
POST /udap/api/pairing HTTP/1.1
Host: 192.168.10.51:8080
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: 105
Connection: Close
User-Agent: Linux/2.6.18 UDAP/2.0 CentOS/5.8

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="pairing">
<name>showKey</name>
</api>
</envelope>

HTTP/1.1 200 OK
Server: Linux/2.6.18-308.4.1.el5 UDAP/2.0 LGSmartTV_33/2.0
Cache-Control: no-store, no-cache, must-revalidate
Date: Tue Jul 17 02:02:39 2012 GMT
Connection: Close
Content-Length: 0
```

Request Pairing (Controller >> Host)

A Controller uses the following XML format when it requests pairing to a Host as HTTP POST.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <api type="pairing">
      <name>hello</name>
      <value>pairing key</value>
      <port>port of the Controller that gets event from host</port>
   </api>
</envelope>
```

A Host that receives the hello request checks the validity of a pairing key and sends a response code for a handling result to the Controller as per the table below:

[Table] Response code of hello request

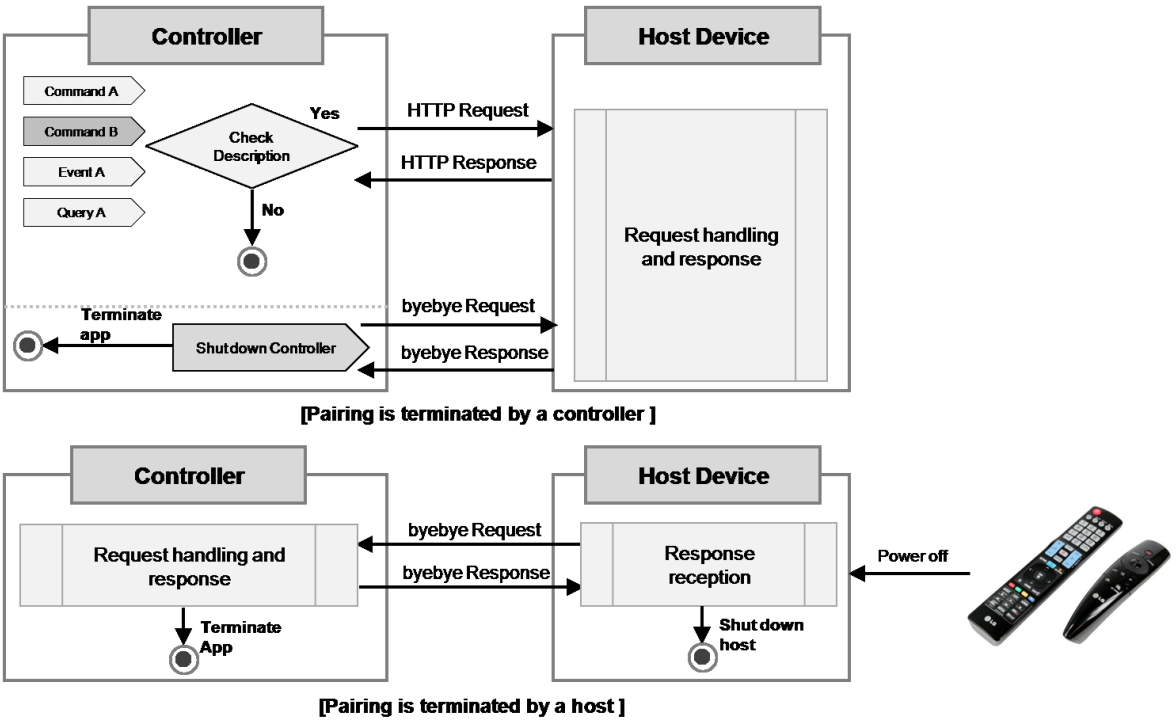
Response Code	Description
200	HTTP/1.1 200 OK The hello request is successful.
400	HTTP/1.1 400 Bad Request The hello request is transmitted in an incorrect format.
401	HTTP/1.1 401 Unauthorized The pairing key value is not valid.
500	HTTP/1.1 500 Internal Server Error During hello request handling, an internal handling error occurs in a Host.
503	HTTP/1.1 503 Service Unavailable Maximum number of Controllers that a Host can accommodate has been exceeded.

If the pairing requested by a Controller is successful, the pop-up window displayed on a Host screen disappears. Otherwise, the pop-up window showing the pairing key displays on the Host screen for another 60 seconds. For better readability of XML, new line characters are included in this document.

```
POST /udap/api/pairing HTTP/1.1
Host: 192.168.10.51:8080
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: 141
Connection: Close
User-Agent: Linux/2.6.18 UDAP/2.0 CentOS/5.8

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="pairing">
<name>hello</name>
<value>166350</value>
<port>8080</port>
</api>
</envelope>

HTTP/1.1 200 OK
Server: Linux/2.6.18-308.4.1.el5 UDAP/2.0 LGSmartTV_33/2.0
Cache-Control: no-store, no-cache, must-revalidate
Date: Tue Jul 17 04:17:37 2012 GMT
Connection: Close
Content-Length: 0
```

End Pairing (Controller <<->> Host)

A Controller must send a byebye request to the Host to end the pairing when it finishes the communication using command/event/query with a Host and terminates its application. Also, a Host sends a byebye request to all the Controllers with which it is communicating when it cannot accept a request from a Controller (for example, when it is turned off by a default TV remote Controller). The diagram below describes how the pairing is terminated by a Controller and a Host.



[Figure] End pairing

Note

When a Host sends a byebye request to a Controller, it does so by using the port number that the Controller has configured in its pairing request XML.

When either a Controller or a Host wants to end pairing, it sends a byebye request as HTTP POST using the following XML format.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <api type="pairing">
      <name>byebye</name>
      <port>port that receives event</port>
   </api>
</envelope>
```

A Controller or a Host that receives the byebye request checks its validity and sends a response code for a handling result to the Controller as the table below:

[Table] Response code of byebye request

Response Code	Description
200	HTTP/1.1 200 OK The byebye request is successful.
400	HTTP/1.1 400 Bad Request The byebye request is transmitted in an incorrect format.
401	HTTP/1.1 401 Unauthorized A response code defined only in a Host. The byebye request is transmitted from a Controller that is not paired.

The actual request and response using the byebye request is as follows. For better readability of XML, new line characters are included in this document.

```
POST /udap/api/pairing HTTP/1.1
Host: 192.168.10.51:8080
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: 121
Connection: Close
User-Agent: Linux/2.6.18 UDAP/2.0 CentOS/5.8

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="pairing">
<name>byebye</name>
<port>8080</port>
</api>
</envelope>

HTTP/1.1 200 OK
Server: Linux/2.6.18-308.4.1.el5 UDAP/2.0 LGSmartTV_33/2.0
Cache-Control: no-store, no-cache, must-revalidate
Date: Tue Jul 17 05:03:31 2012 GMT
Connection: Close
Content-Length: 0
```

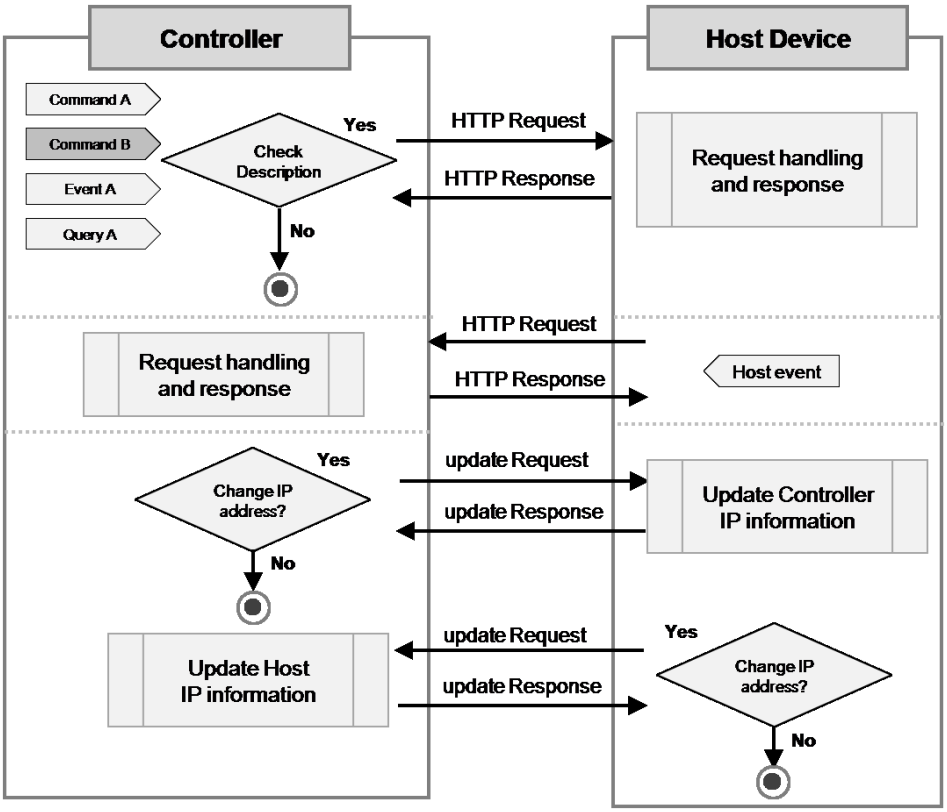
IP Address Change Notification (Controller <<->> Host)

If a Controller and a Host are connected in the DHCP environment, the initially-allocated IP address can be changed if it arrives within the DHCP lease time. If the IP address of a Host is changed, a Controller cannot request command/event/query to the Host anymore. If the IP address of a Controller is changed, the Host cannot send an

event to the Controller. To cope with this situation, the pairing provides an API that can be used to notify the counterpart of the details of the IP address change when their IP addresses are changed.
To notify the IP address change to each other, they send the information as HTTP POST using the following XML format:

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="pairing">
        <name>update</name>
        <value>IP address assigned newly</value>
        <expire> expired IP address </expire>
    </api>
</envelope>
```

The figure below displays the process of the IP address change notification using the update request in diagram form.



[Figure] IP address change notification

A Host or a Controller that receives the update request checks its validity, updates the communication IP address information and sends a response code for a handling result as per the table below:

[Table] Response code of update request

Response Code	Description
200	HTTP/1.1 200 OK The update request is successful.
400	HTTP/1.1 400 Bad Request The update request is transmitted in an incorrect format.
401	HTTP/1.1 401 Unauthorized A response code defined only in a Host. The update request is transmitted from a Controller that is not paired

The actual request and response using the update request is as follows. For better readability of XML, new line characters are included in this document.

```
POST /udap/api/pairing HTTP/1.1
```

```
Host: 192.168.10.51:8080
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: 162
Connection: Close
User-Agent: Linux/2.6.18 UDAP/2.0 CentOS/5.8

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="pairing">
<name>update</name>
<value>192.168.10.51</value>
<expire>192.168.10.50</expire>
</api>
</envelope>

HTTP/1.1 200 OK
Server: Linux/2.6.18-308.4.1.el5 UDAP/2.0 LGSmartTV_33/2.0
Cache-Control: no-store, no-cache, must-revalidate
Date: Tue Jul 17 05:03:31 2012 GMT
Connection: Close
```

Pairing Scenario

A user may find it inconvenient if a Controller needs to receive the pairing key displayed on the screen of a Host whenever it needs to access the Host. To resolve this issue, the following pairing scenario is proposed.

1) When a Controller accesses a Host for the first time, it displays a pairing key on the screen of the Host using the showKey API.

2) When the pairing request of the Controller is successful, the UUID value obtained from the Description and the pairing key of the Host are paired and saved to local storage on the Controller as follows: The UUID and the pairing key value of a Host device are different from those of another Host device and the values can not be changed once they have been created.

[Table] Local storage of a Controller

UUID	Pairing key
UUID #1	Pairing key #1
UUID #2	Pairing key #2
…	…

3) If the pairing key corresponding to the UUID is stored on the local storage of a Controller when the Controller searches for a Host and get the description, the Controller skips the showKey process and completes pairing by reading the stored pairing key value and directly entering into the hello request.

Whether or not to follow the proposed pairing scenario above is up to the Controller application developer who implements the Controller application. **However, it is recommended to implement the scenario for the convenience of users, if possible.**

Command, Event, Query

This chapter provides a description of command, event and query.

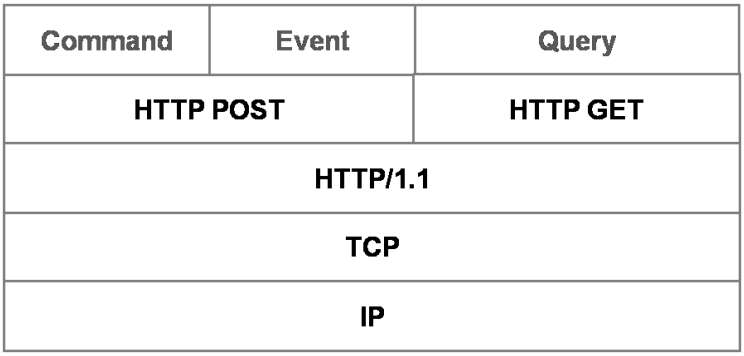
When the pairing request described in **Pairing** is successful, a Controller can request a command/event/query to a Host and receive an event from the Host. To do this, a Controller can perform an HTTP request to access a Host using the IP address and the port number obtained from Discovery and Description. A Host sends an event to a Controller using the port number specified in the XML that the Controller sends in its pairing request.

Each one of command, event, or query is based on XML and performs the following function.

[Table] Roles of Command, event, and query

Category	Role
Command	A Controller sends a specific command to a Host to control the Host.
Event	A Controller or a Host sends an event to each other when their status is changed or a specific event occurs.
Query	A Controller uses a query to get specific data from a Host.

The protocol stack used by a command, event or query is as follows:



[Figure] Protocol stacks of command, event, and query

For details of API call methods on each service of a command, event, or query, refer to LG UDAP 2.0 Service Profiles.

Note

When building XML, avoid inserting the line change character (“\r”, “\n”) if possible. This reduces data transmission size by excluding unnecessary characters and speeds up the operation at the XML parser of a Host.

This chapter includes the following sections.

Common Rules

The command, event and query have the following common rules.

1) The character encoding used for communication is UTF-8.

2) If an XML data value has a special character, encode the character before transmission as shown in the following table:

[Table] Encoding XML special characters

Special Character	Encoding Value
\n (new line character)
\	'
“	"
	<

Special Character	Encoding Value
>	&gt;
&	&amp;

For example, the string **<NEWS & 24>** must be encoded as **<NEWS & 24>** to include in XML document.

3) A Controller and a Host use the following format of USER-AGENT header.
User-Agent: ***OS/version*** **UDAP/2.0** ***product/version***

4) The HTTP server of a Host supports the HTTP persistent connection as well as general HTTP communication (Connection: Close). Therefore, it can handle continuous requests of a command, event or query at a high speed. The process for a Controller to handle each communication is as follows:

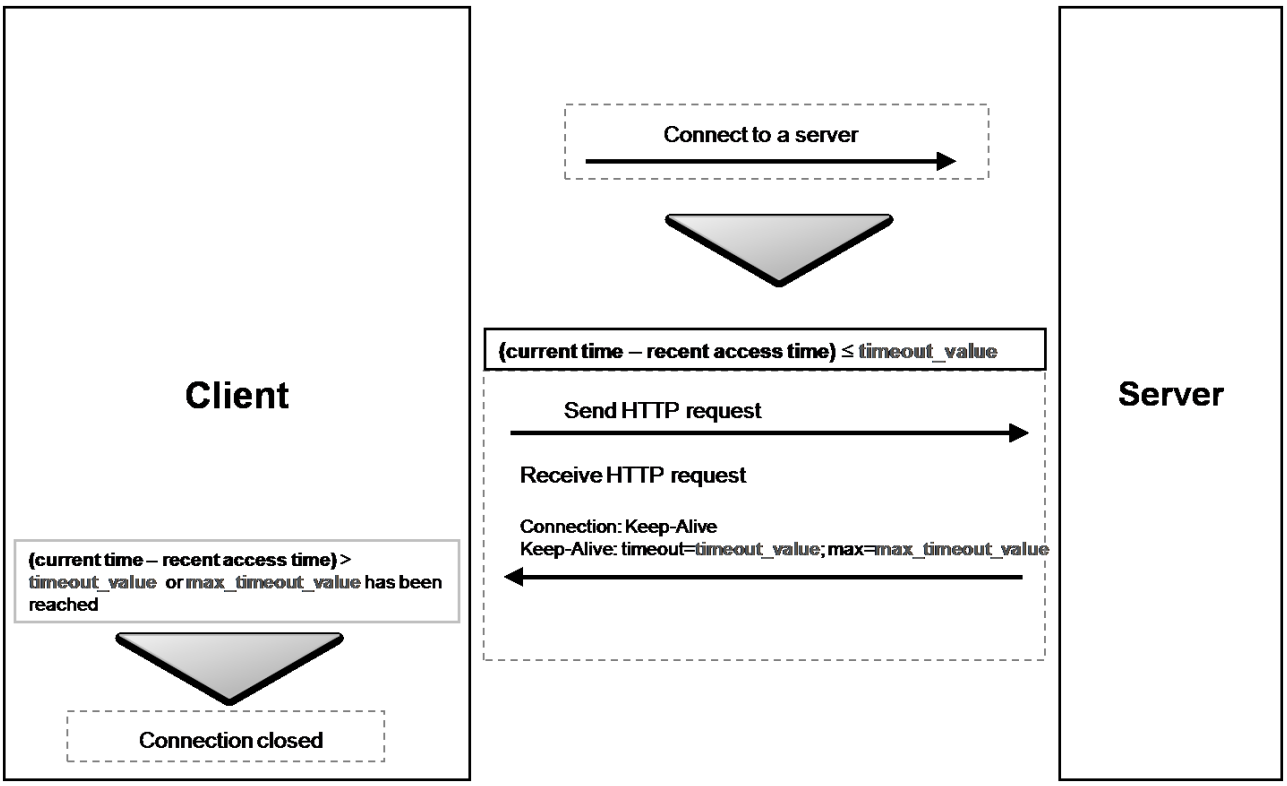
[General HTTP communication]

1. Connect to a target server.
2. Set the value of a connection header to Close and send a request such as HTTP GET or HTTP POST, etc.
3. Receive the response of a server for the request.
4. Close the connection to the server.

[HTTP persistent connection]

1. Connect to a target server.
2. Set the value of a connection header to Keep-Alive and send a request such as HTTP GET or HTTP POST, etc.
3. Receive the response from a server for the request and check if the response header of server includes "Connection: Keep-Alive" and "Keep-Alive: timeout=timeout_value; max=max_timeout_value".
4. If the server response includes two headers described in 3, the connection to the server can be maintained during the set timeout (in seconds) without disconnection.
   - The connection to the server can be maintained if the value of "current time - recent access time" is the same or smaller than timeout_value.
   - Even when the value of "current time - recent access time" is the same or smaller than timeout_value, the connection to the server is not valid if the max_timeout_value has been reached.
5. If the value of "current time - recent access time" is greater than timeout_value or the max_timeout_value has been reached, the connection to the server is closed.

The figure below displays an HTTP persistent connection in diagram form.



[Figure] Conceptual diagram of an HTTP persistent connection

The following examples show two types of HTTP communication. The equivalent rules are also applied to Event, Query.

```
POST /udap/api/command HTTP/1.1
Host: target_ip:port
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: data length to transfer
Connection: Keep-Alive
User-Agent: OS/version UDAP/2.0 product/version

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="command">
<name>apiName</name>
<paramName>param value</paramName>
<!-- additional parameters are listed if available -->
</api>
</envelope>

HTTP/1.1 200 OK
SERVER: OS/version UDAP/2.0 product/version
Cache-Control: no-store, no-cache, must-revalidate
Date: Time when the host response is occurred
Connection: Keep-Alive
Keep-Alive: timeout=timeout_value, max=max_timeout_value
Content-Length: 0
```

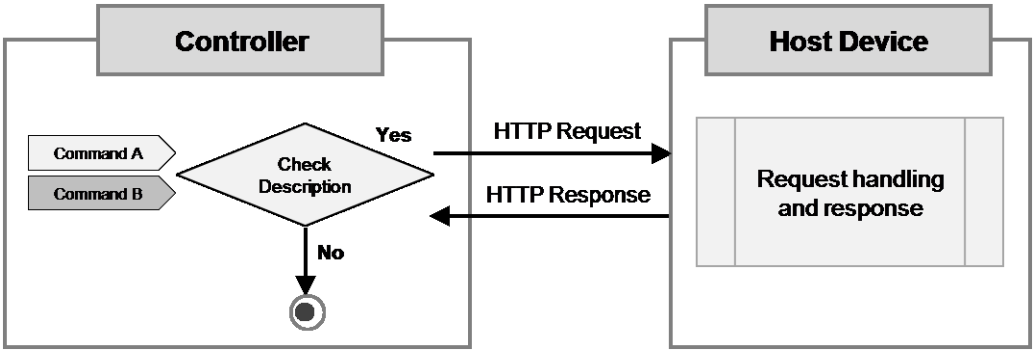
```
POST /udap/api/command HTTP/1.1
Host: target_ip:port
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: data length to transfer
Connection: Close
User-Agent: OS/version UDAP/2.0 product/version

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="command">
<name>apiName</name>
<paramName>param value</paramName>
<!-- additional parameters are listed if available -->
</api>
</envelope>

HTTP/1.1 200 OK
SERVER: OS/version UDAP/2.0 product/version
Cache-Control: no-store, no-cache, must-revalidate
Date: Time when the host response is occurred
Connection: Close
Content-Length: 0
```

Command (Controller >> Host)

A command complies with all the Common Rules and is sent as HTTP POST using the following path.

http://target_ip:port/udap/api/command

A command is used only when a Controller controls a Host. There is no command sent from a Host. The command's call is uni-directional between a Controller and a Host as shown below:



[Figure] Call direction of a command

The XML format used in a command uses the following common format regardless of service type. The black font is a fixed value and the red Italic font can vary per command. For details of command call methods on each service, refer to LG UDAP 2.0 Service Profiles.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="command">
        <name>apiName</name>
        <paramName>param value</paramName>
        <!-- additional parameters are listed if available -->
    </api>
</envelope>
```

To increase the response speed for a command execution result, UDAP uses an HTTP response code to notify the Controller of the command execution result. It does not include a separate HTTP response body. The value of a response code that can be used to display a command execution result as an HTTP response code is shown in the following table:

[Table] Response code of a command

Response code	Description
200	HTTP/1.1 200 OK The command execution is successful.
400	HTTP/1.1 400 Bad Request The command format is not valid or it has an incorrect value.
401	HTTP/1.1 401 Unauthorized A command is sent when a Host and a Controller are not paired.
404	HTTP/1.1 404 Not Found The POST path of a command is incorrect.
500	HTTP/1.1 500 Internal Server Error The command execution has failed.

The format of the request and the response using a command is as follows: For better readability of XML, new line characters are included in this document. For a continuous operation (for example, mouse movement), a command can be sent out quickly by using an HTTP persistent connection.

```
POST /udap/api/command HTTP/1.1
Host: host_ip:port
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: byte length of Command XML
Connection: Close
User-Agent: OS/version UDAP/2.0 product/version

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="command">
<name>apiName</name>
<paramName>param value</paramName>
<!-- additional parameters are listed if available -->
</api>
</envelope>

HTTP/1.1 200 OK
Server: OS/version UDAP/2.0 product/version
Cache-Control: no-store, no-cache, must-revalidate
Date: Time when the host response is occurred
Connection: Close
Content-Length: 0
```

Event (Controller <<->> Host)

An event complies with all the common rules in Common Rules and is sent as HTTP POST using the following path.

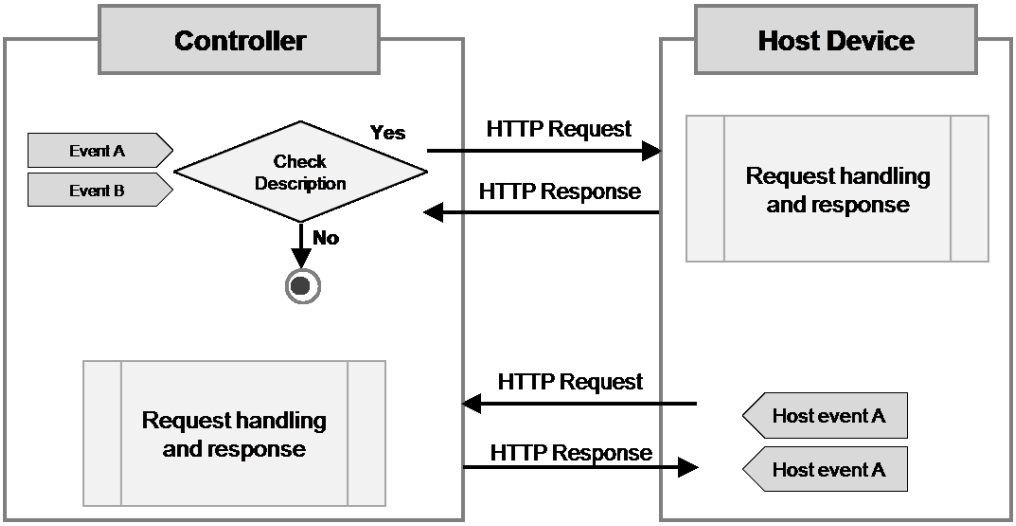
http://target_ip:port/udap/api/event

A Controller or a Host sends an event to each other when their status is changed or a specific event occurs. An event

can be transmitted in the following cases.

1) A Controller sends an event to a Host when its status has changed or a specific event occurs.
2) A Host sends an event to a Controller with which it is paired when the Host's status has changed or a specific event occurs.
3) A Controller or a Host sends an event to each other when their status has changed or a specific event occurs.

An event is bi-directional between a Controller and a Host, Controller --> Host, Host --> Controller, Controller <--> Host, as shown below. If an event is received to a Controller from a Host which is not requested by the Controller, the event is ignored.



[Figure] Transmission direction of event

The XML format used in an event uses the following common format regardless of service type. The black font is a fixed value and the red Italic font can vary per event. **For details of event transmission methods on each service, refer to LG UDAP 2.0 Service Profiles.**

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>apiName</name>
        <paramName>param value</paramName>
        <!-- additional parameters are listed if available -->
    </api>
</envelope>
```

To increase the response speed for an event processing result, UDAP uses an HTTP response code to notify the counterpart of the event processing result. It does not include a separate HTTP response body. The value of a response code that can be used to display an event processing result as an HTTP response code is shown in the following table:

[Table] Response code of an event

Response code	Description
200	HTTP/1.1 200 OK The event execution is successful.
400	HTTP/1.1 400 Bad Request The event format is not valid or it has an incorrect value.
401	HTTP/1.1 401 Unauthorized An event is sent when a Host and a Controller are not paired.
404	HTTP/1.1 404 Not Found The POST path of an event is incorrect.
500	HTTP/1.1 500 Internal Server Error Event Execution Failure

The request and the response formats of an event are as follows. For better readability of XML, new line characters

are included in this document. When the events are generated continuously, it can send the events quickly using an HTTP persistent connection.

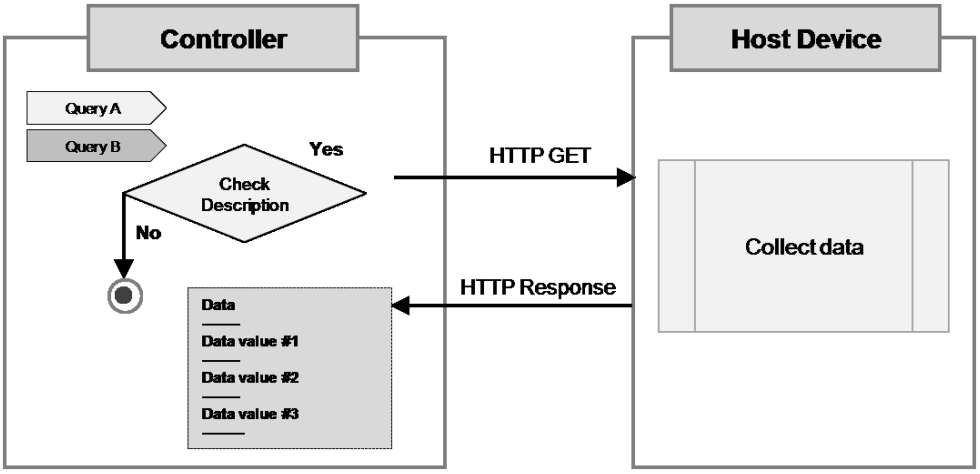
```
POST /udap/api/command HTTP/1.1
Host: host_ip:port
Cache-Control: no-cache
Content-Type: text/xml; charset=utf-8
Content-Length: Byte length for Command XML
Connection: Close
User-Agent: OS/version UDAP/2.0 product/version

<?xml version="1.0" encoding="utf-8"?>
<envelope>
<api type="event">
<name>apiName</name>
<paramName>param value</paramName>
<!-- additional parameters are listed if available -->
</api>
</envelope>

HTTP/1.1 200 OK
Server: OS/version UDAP/2.0 product/version
Cache-Control: no-store, no-cache, must-revalidate
Date: Time when the host response is occurred
Connection: Close
Content-Length: 0
```

Note

Some services may use a command or an event in the JSON format rather than the XML format. This kind of service does not use the POST path described above. Instead, it uses a POST path that the service defined separately. Refer to the App To App services in LG UDAP 2.0 Service Profiles.

Query (Controller >> Host)

A query complies with all the common rules in Common Rules and a Controller requests information to a Host using HTTP GET. The query's call is uni-directional between a Controller and a Host as shown below:



[Figure] Call direction of a query

Query includes URL encoding rule in addition to the common rules described in Common Rules. This rule is used in internet standard HTTP/1.1 and not defined in UDAP separately. Refer to Annex A URL Encoding Reference for detailed information.(the string **<NEWS & 24>** is encoded as **%3cNEWS%20%26%2024%3e)**

The URL format used in a query uses the following common format regardless of service type. The black font is a fixed value and the red Italic font can vary per event.

```
http://target_ip:port/udap/api/data?target=apiName&paramName1=URL_Encode(param_value1)&paramName2=URL_Encode(param_value2)
```

In the above URL, the paramName and its corresponding param_value can vary per query type. There is a query that can be called just with apiName without the paramName and param_value. **For details of query call methods on each service, refer to LG UDAP 2.0 Service Profiles.**

When a Controller requests a query to a Host, the Host analyzes the API name and parameter of the query, collects data for the request, and creates and returns a response to the Controller in the common XML format. For a query call, the possible value of a response code is as follows:

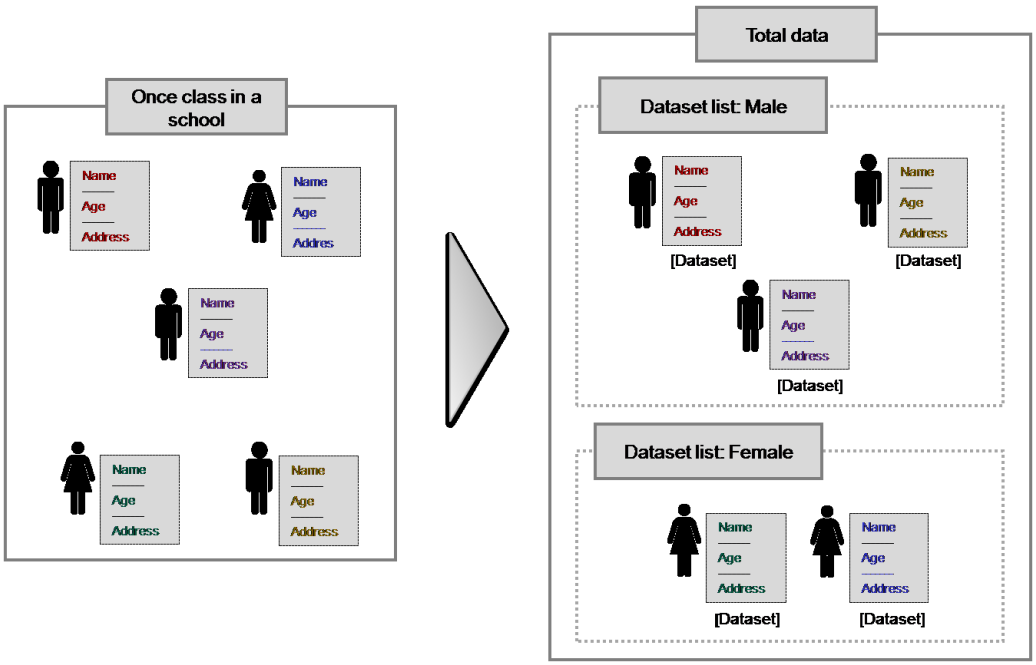
[Table] Response codes of a query

Response code	Description
200	HTTP/1.1 200 OK The query execution is successful.
401	HTTP/1.1 401 Unauthorized A query is sent when a Host and a Controller are not paired.
404	HTTP/1.1 404 Not Found The request path of a query is incorrect.
500	HTTP/1.1 500 Internal Server Error Query execution failure

The XML format that is delivered to a Controller as the response to a query uses the following common format regardless of service type and the Content-Type is "text/xml; charset=utf-8". UDAP uses the following concept to generalize the response XML format of a query.

1) A dataset can have several values.
2) A dataset list can have several datasets.
3) The total query response data can have several dataset lists.

To help you understand the above concept, let's assume we group the students in one class of a school. There are male and female students in a class and they have their own information such as name, age, and address. Here, if we assume one **dataset** represents one student's information including name, age, and address, a **dataset list** has these **datasets** for each gender (male and female) and it means that all the students in the class are included in the two **dataset lists**. For a better understanding, the diagram for this concept is as follows:



[Figure] Example of response XML configuration of a query

Based on this, the response XML format of a query is defined as follows. "data" means dataset, "dataList" means

dataset list, and the name attribute of dataList means the name of dataList. (Value corresponding to "Male" and "Female" in the above diagram)

In the below XML format, the black font is a fixed value and the red Italic font can vary per query. The Controller receives the value by decoding because the values in the below XML format can be sent with URL-encoded. **For details of the response XML format of a query for each service, refer to LG UDAP 2.0 Service Profiles.**

Response XML Elements of a Query

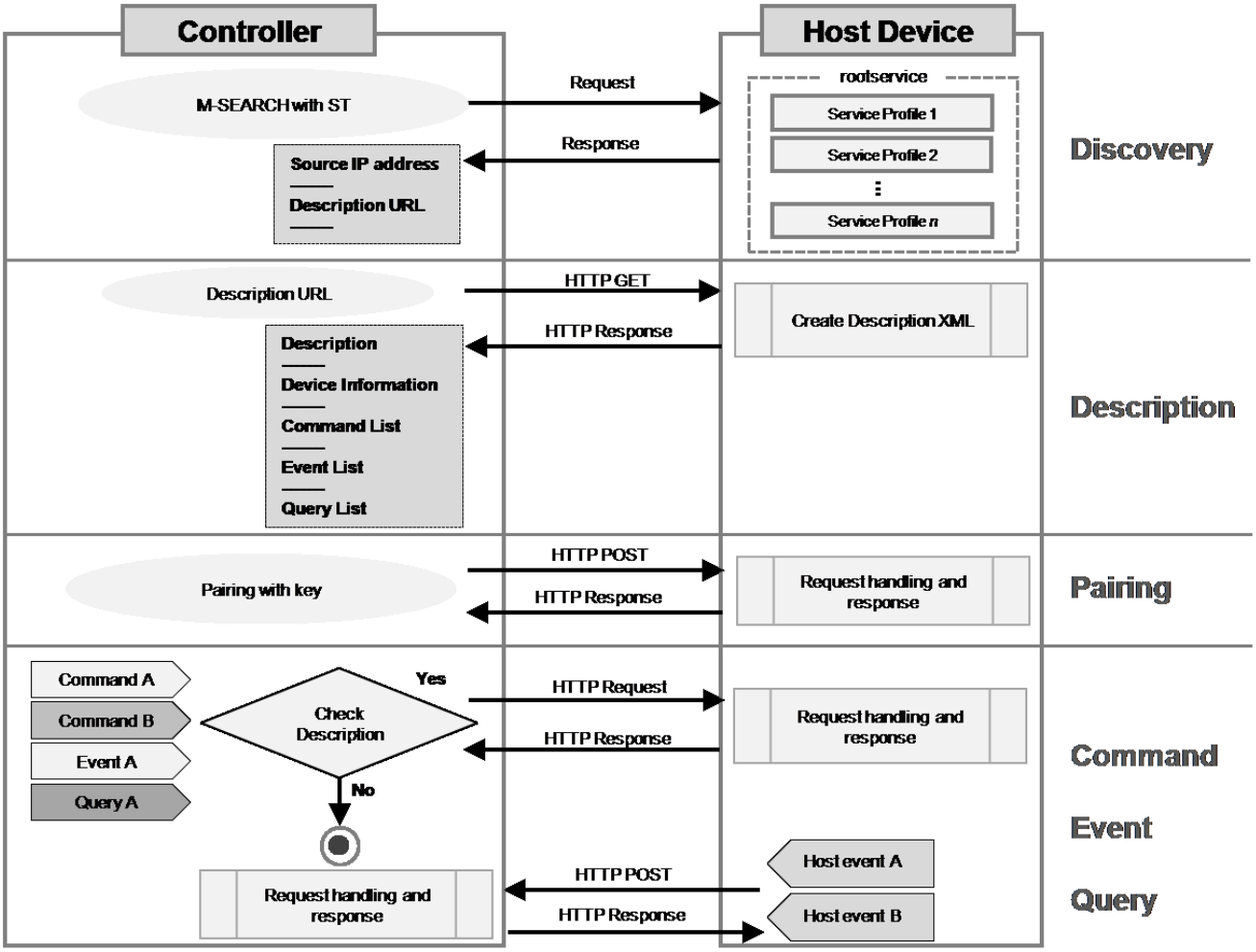
```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <device>
   <dataList name="data_list_name">
      <data>
         <valueName>value</valueName>  <!-- value can be URL-encoded -->
         <valueName>value</valueName>  <!-- value can be URL-encoded -->
         <!-- additional data values are listed if available -->
      </data>
      <data>
         <valueName>value</valueName>  <!-- value can be URL-encoded -->
         <valueName>value</valueName>  <!-- value can be URL-encoded -->
         <valueName>value</valueName>  <!-- value can be URL-encoded -->
         <!-- additional data values are listed if available -->
      </data>
      <data>
         <valueName>value</valueName>  <!-- value can be URL-encoded -->
         <!-- additional data values are listed if available -->
      </data>
   </dataList>
   <dataList name="data_list_name">
   <!-- additional data is listed if available -->
   </dataList>
   <!-- additional data lists are listed if available -->
</envelope>
```

Note

Some services may use a command or an event in the JSON format rather than the XML format. This kind of service does not use the POST path described above. Instead, it uses a POST path that the service defined separately. Refer to the App To App service in LG UDAP 2.0 Service Profiles.

Examples of Controller Application Operation

This chapter displays the examples of a Controller application communicating with a Host at each step of UDAP in diagram form.



[Figure] Overall operation scenario of a Controller

Annex A URL Encoding Reference

The following table shows URL-encoding value for each ASCII character. The URL-encoding is needed because HTTP URL format supposes only ASCII value by default. Since the internet has been getting globalized, URL-encoding with hexadecimal was suggested from W3C for representing any character in HTTP URL, which is not representable as ASCII value, and now it is used as internet standard.
For example, & is represented as %26 and white space is represented as %20, therefore, the string NEWS & 24 is encoded as NEWS%20%26%2024

[Table] URL encoding value

ASCII character	URL-encoding
space	%20
!	%21
"	%22
#	%23
$	%24
%	%25
&	%26

ASCII character	URL-encoding
'	%27
(	%28
)	%29
*	%2A
+	%2B
,	%2C
-	%2D
.	%2E
/	%2F
:	%3A
;	%3B
<	%3C
=	%3D
>	%3E
?	%3F
@	%40
[	%5B
\	%5C
]	%5D
^	%5E
_	%5F
`	%60
{	%7B
|	%7C
}	%7D
~	%7E
`	%80
‚	%82
ƒ	%83
„	%84
…	%85
†	%86

ASCII character	URL-encoding
‡	%87
ˆ	%88
‰	%89
Š	%8A
‹	%8B
Œ	%8C
Ž	%8E
‘	%91
’	%92
“	%93
”	%94
•	%95
–	%96
—	%97
˜	%98
™	%99
š	%9A
›	%9B
œ	%9C
ž	%9E
Ÿ	%9F
¡	%A1
¢	%A2
£	%A3
¤	%A4
¥	%A5
¦	%A6
§	%A7
¨	%A8
©	%A9
ª	%AA
«	%AB

ASCII character	URL-encoding
¬	%AC
®	%AE
¯	%AF
°	%B0
±	%B1
²	%B2
³	%B3
´	%B4
µ	%B5
¶	%B6
·	%B7
¸	%B8
¹	%B9
º	%BA
»	%BB
¼	%BC
½	%BD
¾	%BE
¿	%BF
À	%C0
Á	%C1
Â	%C2
Ã	%C3
Ä	%C4
Å	%C5
Æ	%C6
Ç	%C7
È	%C8
É	%C9
Ê	%CA
Ë	%CB
Ì	%CC

ASCII character	URL-encoding
Í	%CD
Î	%CE
Ï	%CF
Ð	%D0
Ñ	%D1
Ò	%D2
Ó	%D3
Ô	%D4
Õ	%D5
Ö	%D6
×	%D7
Ø	%D8
Ù	%D9
Ú	%DA
Û	%DB
Ü	%DC
Ý	%DD
Þ	%DE
ß	%DF
à	%E0
á	%E1
â	%E2
ã	%E3
ä	%E4
å	%E5
æ	%E6
ç	%E7
è	%E8
é	%E9
ê	%EA
ë	%EB
ì	%EC

ASCII character	URL-encoding
í	%ED
î	%EE
ï	%EF
ð	%F0
ñ	%F1
ò	%F2
ó	%F3
ô	%F4
õ	%F5
ö	%F6
÷	%F7
ø	%F8
ù	%F9
ú	%FA
û	%FB
ü	%FC
ý	%FD
þ	%FE
ÿ	%FF

LG UDAP 2.0 Service Profiles

This document describes the actual operation of the service utilizing the Universal Discovery & Access Protocol (UDAP) and the details of the API.

Service Profile Overview

This chapter provides the overview of the service profiles implemented using the UDAP.

This chapter includes the following sections.

Types and Roles of Service Profiles

The UDAP implements various applications which operate on the Host device, based on the common base protocol which is described in LG UDAP 2.0 Protocol Specification.

The Host (LG Smart TV) provides access to the service profiles listed in the table below and functions as well.

[Table] All service profiles

Service profile	Function and role
Remote Controller Service (netrcu)	Provides query for control functions of the Host remote Controller and the magic Controller (command, event) and the TV information (e.g., channels list, current channel info, …).
Text Input Service (smartText)	When the web browser on the Host or a user app is in the UI mode for text input, this service allows the Controller to send text input to the Host or notify that text input is available.
Mobile Home Service (mobilehome)	Fetches the list of currently installed apps from the Host and provides app functions, such as remote run and end, and other control functions.
App To App Service (AppToApp)	Helps a specific app on the Host and a specific app on the Controller are intimately linked and allows the developer to implement a specialized application of functions of a specific app.

Note

Since events of UDAP 2.0 service profiles dynamically occur according to the operation status of the Host, the Controller must be able to ignore any incoming events from the Host which are not supposed to be processed by it. If an application is written for the Controller by mingling APIs of multiple service profiles, a search should be performed with **udap:rootservice** to obtain descriptions of all service profiles.

Descriptions of All Service Profiles

As described in LG UDAP 2.0 Protocol Specification, the descriptions of all service profiles supported by LG Smart TV can be obtained by entering **udap:rootservice** as the ST of the SSDP.

Remote Controller Service (netrcu)

This chapter describes on Remote Controller service that provides query for control functions of the Host remote

Controller and the magic Controller (command, event) and the TV information (e.g., channels list, current channel info, …).

This chapter includes the following sections.

Discovery & Description

The ST value for searching the Remote Controller service is as follows.

```
urn:schemas-udap:service:netrcu:1
```

If searching the SSDP with the ST value above, the descriptions are returned as below. The device section of the description XML contains the model name and the UUID value of the actual Host.

```
<?xml version="1.0" encoding="utf-8"?>
▼<envelope>
  ▼<device>
      <deviceType>TV</deviceType>
      <modelName>LGSmartTV_33</modelName>
      <friendlyName>LG Smart TV</friendlyName>
      <manufacturer>LG Electronics</manufacturer>
      <uuid>0000004c-0600-4c04-0e04-0290040e900e</uuid>
    </device>
    <port>8080</port>
  ▼<serviceList>
    ▼<service name="netrcu">
      ▼<apiList>
        ▼<api type="command">
            <name>HandleChannelChange</name>
            <name>HandleTouchWheel</name>
            <name>HandleTouchClick</name>
            <name>HandleTouchMove</name>
            <name>HandleKeyInput</name>
          </api>
        ▼<api type="event">
            <name>3DMode</name>
            <name>DragMode</name>
            <name>CallStateChanged</name>
            <name>ChannelChanged</name>
            <name>CursorVisible</name>
          </api>
        ▼<api type="query">
            <name>volume_info</name>
            <name>is_3d</name>
            <name>screen_image</name>
            <name>channel_list</name>
            <name>context_ui</name>
            <name>cur_channel</name>
          </api>
        </apiList>
      </service>
    </serviceList>
</envelope>
```

Note

Since **udap:rootservice** searchs all services existing on Host, it can also be used by Remote Controller service and descriptions in Descriptions of All Service Profiles can be obtained.

Command (Controller >> Host)

Commands of the Remote Controller service complies with all common rules in the LG UDAP 2.0 Protocol Specification and are sent in the HTTP POST method using the path below.

http://target_ip:port/udap/api/command

The XML format used for the command is the common format defined in LG UDAP 2.0 Protocol Specification. Black fonts indicate fixed values, and red italic fonts indicate values variable depending on commands.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <api type="command">
      <name>apiName</name>
      <paramName>param value</paramName>
      <!-- Additional parameters are listed if available -->
   </api>
</envelope>
```

Upon successful execution of a command, HTTP/1.1 200 OK is returned. For details of response codes, refer to LG UDAP 2.0 Protocol Specification.

Commands are as follows:
- HandleKeyInput
- HandleTouchMove
- HandleTouchClick
- HandleTouchWheel
- HandleChannelChange

HandleKeyInput

This command transfers the virtual remote Controller key code value of the Host. After receiving this virtual remote Controller key code value, the Host converts the value into an actual Host key code and performs the key input action.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <api type="command">
      <name>HandleKeyInput</name>
      <value>Value key code of remote Controller</value>
   </api>
</envelope>
```

[Table] Parameters of HandleKeyInput

Parameter	Description
Value	Name of the XML element to which the virtual key code value of the Host is assigned. For details of virtual remote Controller key code values, see the virtual key code table in Annex A.

[Table] Actions of HandleKeyInput

Item	Description
Persistent connection status	In case of sending a single key code value to the Host, use Connection: Close. To send multiple values continuously and quickly to the Host, enable the persistent connection by using Connection: Keep-Alive. For details of persistent connection, refer to LG UDAP 2.0 Protocol Specification.
Related events	If the Host successfully processes HandleKeyInput, the event below is sent to the Controller. ChannelChanged event of the **netrcu** service.

HandleTouchMove

This command controls the movement of the magic remote Controller cursor of the Host on the screen. After receiving this, the Host moves the mouse cursor on the screen according to the X and Y coordinate values specified by the Controller.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="command">
        <name>HandleTouchMove</name>
        <x> Displacement value of X coordinate </x>
        <y> Displacement value of Y coordinate </y>
    </api>
</envelope>
```

[Table] Parameters of HandleTouchMove

Parameter	Description
x	Displacement value for the X coordinate. Using the current X coordinate as a reference, a negative value denotes the movement to the left and a positive value denotes the movement to the right.
y	Displacement value for the Y coordinate. Using the current Y coordinate as a reference, a negative value denotes an upward movement and a positive value denotes a downward movement.

For details of mouse coordinates movement, see the diagram below.

[Table] Actions of HandleTouchMove

Item	Description
Persistent connection status	Since the action takes places very quickly, enable the persistent connection by using Connection: Keep-Alive. For details of persistent connection, refer to LG UDAP 2.0 Protocol Specification.
Related events	Before sending HandleTouchMove, send the CursorVisible event to the Host so that the mouse cursor appears on the Host screen first. CursorVisible event of the **netrcu** service.

[Figure] Calculation of Displacement Value of Mouse Coordinates

HandleTouchClick

The HandleTouchClick event sends the mouse click command while the magic remote Controller cursor of the Host

is stationary at a specific position on the screen. After receiving this, the Host generates the click command at its current screen position.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <api type="command">
      <name>HandleTouchClick</name>
   </api>
</envelope>
```

HandleTouchClick has no additional parameters.

[Table] Actions of HandleTouchClick

Item	Description
Persistent connection status	Use Connection: Close when sending a single click to the Host. To send multiple clicks continuously and quickly to the Host, enable the persistent connection by using Connection: Keep-Alive. For details of persistent connection, refer to LG UDAP 2.0 Protocol Specification.
Related events	Before sending HandleTouchClick, send the CursorVisible event to the Host so that the mouse cursor appears on the Host screen first. After receiving HandleTouchClick, if the Host is not currently showing the cursor, the Host only shows the cursor on its screen and performs no click action. CursorVisible event of the **netrcu** service.

HandleTouchWheel

The HandleTouchWheel event sends the command to perform the wheel function of the magic remote Controller of the Host. After receiving this command, the Host generates the upward/download wheel movement at its current screen position. When on any other screen with scrolls (for example, a web browser screen), the action is performed in the same way as the mouse wheel on the PC. When the wheel action is sent on the TV playback screen, the Host triggers the channel up/down action.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <api type="command">
      <name>HandleTouchWheel</name>
      <value> Up/down scrolling direction </value>
   </api>
</envelope>
```

[Table] Parameters of HandleTouchWheel

Parameter	Description
value	This specifies upward or downward direction of the scroll movement. The following two values are available. up: Scroll up down: Scroll down

[Table] Acrions of HandleTouchWheel

Action	Description
Persistent connection status	Use Connection: Close when sending a wheel action to the Host. To send multiple wheel actions continuously and quickly to the Host, enable the persistent connection by using Connection: Keep-Alive. For details of persistent connection, refer to LG UDAP 2.0 Protocol Specification.
Related events	Before sending HandleTouchWheek, send the CursorVisible event to the Host so that the mouse cursor appears on the Host screen first. CursorVisible event of the **netrcu** service.

HandleChannelChange

This command directly changes the channel by using the TV channels list information of the Host instead of using the channel up/down keys on the remote Controller. As the parameters used in HandleChannelChange make use of the channels list information, see the channels list for details.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="command">
        <name>HandleChannelChange</name>
        <major>major number of the channel</major>
        <minor>minor number of the channel</minor>
        <sourceIndex>source index of the channel</sourceIndex>
        <physicalNum>physical number of the channel</physicalNum>
    </api>
</envelope>
```

[Table] Parameters of HandleChannelChange

Parameter	Description
major	This is the major channel number which can be obtained from the channels list information.
minor	This is the minor channel number which can be obtained from the channels list information.
sourceIndex	This is the channel source index which can be obtained from the channels list information.
physicalNum	This is the physical channel number which can be obtained from the channels list information.

For example, in case of the 11-1 MBC DTV channel, the major number is 11 and the minor number is 1. Its source index and physical number are determined by the values in the channels list information of the Host.

[Table] Actions of HandleChannelChange

Item	Description
Persistent connection status	Use Connection: Close when sending a single channel change to the Host. To send multiple channel changes continuously and quickly to the Host, enable the persistent connection by using Connection: Keep-Alive. For details of persistent connection, refer to LG UDAP 2.0 Protocol Specification.
Related events and queries	Channels list query of the **netrcu** service: HandleChannelChange has the parameter information to be used. ChannelChanged event of the **netrcu** service: Sent to the Controller when the channel change is successful.

Event (Controller <<->> Host)

Events of the Remote Controller service complies with all common rules in the LG UDAP 2.0 Protocol Specification and are sent in the HTTP POST method using the path below.

http://target_ip:port/udap/api/event

The XML format used for the event is the common format defined in the LG UDAP 2.0 Protocol Specification. Black fonts indicate fixed values, and red italic fonts indicate values variable depending on events.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>apiName</name>
        <paramName>param value</paramName>
        <!-- Additional parameters are listed if available -->
   </api>
</envelope>
```

Upon successful handling of an event, HTTP/1.1 200 OK is returned. For details of response codes, refer to LG

UDAP 2.0 Protocol Specification.

Events are as follows:
- CursorVisible (Controller <<->> Host)
- ChannelChanged (Controller << Host)
- CallStateChanged (Controller >> Host)
- DragMode (Controller >> Host)
- 3DMode (Controller << Host)

CursorVisible (Controller <<->> Host)

This event notifies the Host that the Controller is ready to use the mouse cursor. Reversely, the event also notifies the Controller that the mouse cursor of the magic remote Controller has become available for use.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>CursorVisible</name>
        <value>true or false</value>
        <mode>auto</mode>
    </api>
</envelope>
```

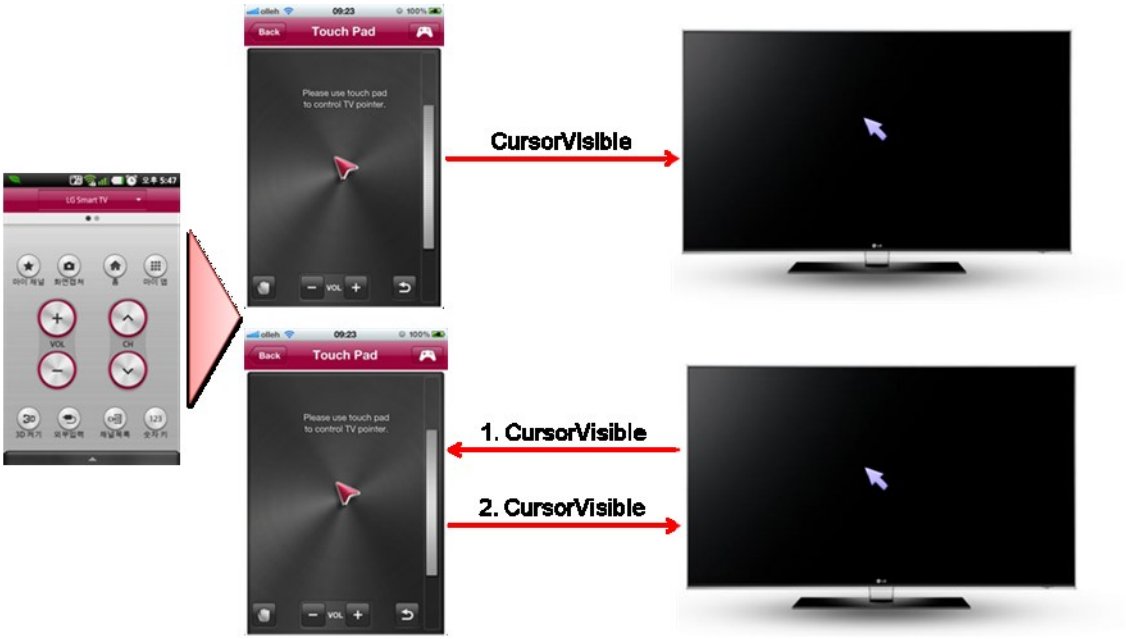
When the Controller enters the UI mode for using the mouse cursor, it notifies the Host of it so that the mouse cursor is displayed on the Host. Reversely, when notified by the Host that the magic remote Controller mouse cursor is available for use, the Controller switches into the UI mode for using the mouse cursor and then sends the CursorVisible event to the Host so that the Host displays the magic mouse cursor. The diagram below shows an overview of this operation.

[Table] Parameters of CursorVisible

Parameter	Description
value	Shows or hides the mouse cursor. The following two values are available. **true**: Show mouse cursor **false**: Hide mouse cursor
mode	This element is for showing the mode which displays the mouse cursor and currently not in use. It is fixed as auto.

[Table] Actions of CursorVisible

Item	Description
Persistent connection status	Since this event does not occur frequently, use Connection: Close.
Related commands	None



[Figure] Making Mouse Cursor Visible

ChannelChanged (Controller << Host)

When a channel change related command of the Controller is successfully processed by the Host, the changed channel details are sent to the Controller.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>ChannelChanged</name>
        <labelName>label name of the current source</labelName>
        <inputSourceType>source type</inputSourceType>
        <inputSourceName>source name</inputSourceName>
        <audioCh>audio channel or not</audioCh>
        <progName>program name</progName>
        <chtype>channel transferring type</chtype>
        <major>physical major number of the channel</major>
        <minor> physical minor number of the channel </minor>
        <sourceIndex>source index of the channel</sourceIndex>
        <physicalNum>physical number of the channel</physicalNum>
        <chname>channel name</chname>
    </api>
</envelope>
```

When the Controller enters the UI mode for using the mouse cursor, it notifies the Host of it so that the mouse cursor is displayed on the Host. Reversely, when notified by the Host that the magic remote Controller mouse cursor is available for use, the Controller switches into the UI mode for using the mouse cursor and then sends the CursorVisible event to the Host so that the Host displays the magic mouse cursor. The diagram below shows an overview of this operation.

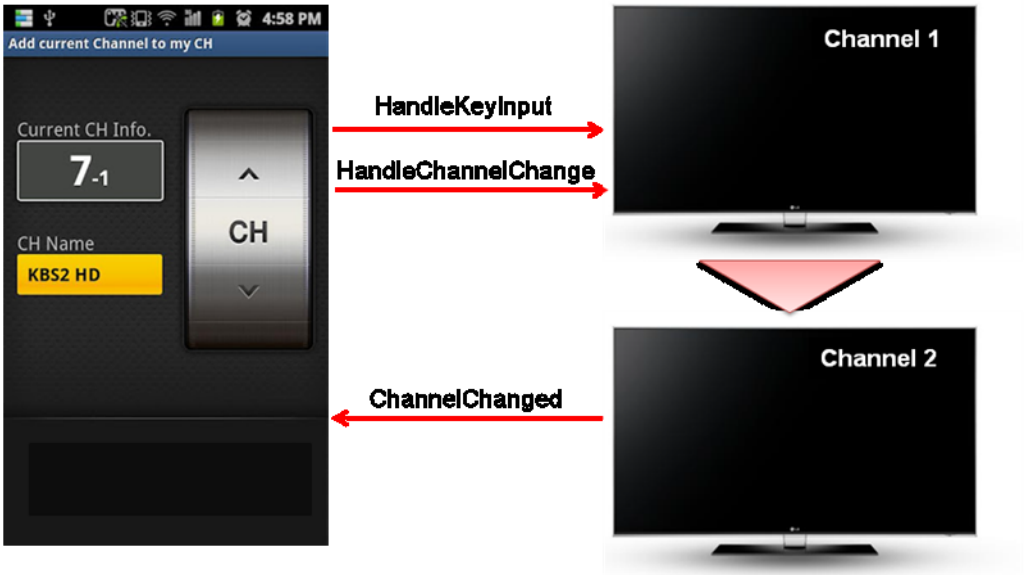
[Table] Parameters of ChannelChanged

Parameter	Description
labelName	Label name of the current input source
inputSourceType	Type of the current input source. The following values are available. **0**: TV, **1**: External input, **2**: RGB
inputSourceName	Name of the current input source. The following values are available. **Antenna**: Input via the TV antenna **HDMI**: HDMI input. The actual value has the label number for the HDMI input port of the TV at the end (for example, HDMI1, HDMI2, …). **Component**: Component input. The actual value has the label number for the

Parameter	Description
	component input port of the TV at the end (for example, Component1, Component2, …) **Composite**: Composite input. The actual value has the label number for the composite input port of the TV at the end (for example, Composite1, Composite2, …) **RGB**: RGB-PC input
audioCh	Shows whether the current channel is an audio channel. The following values are available. **0**: Normal channel **1**: Audio channel
progName	Program name currently on the air on the current channel
chtype	Type of the current channel. The following values are available. **satellite**: Satelite broadcast **cable**: Cable broadcast **terrestrial**: Terrestial braodcast
major	Physical major number of the current channel
minor	Physical minor number of the current channel
sourceIndex	Source index of the current channel
physicalNum	Physical number of the current channel
chname	Name of the current channel

[Table] Actions of ChannelChanged

Item	Description
Persistent connection status	Transferred to the Controller with Connection: Close.
Related commands	This event is generated when changing channels. For details, see the descriptions of HandleKeyInput and HandleChannelChange.



[Figure] Generation of ChannelChanged Event

CallStateChanged (Controller >> Host)

If the Controller is a smartphone, there may be an incoming call while using an app. In this case, the Host is notified of the incoming call so that the Host can perform additional processes for the call-related events of the Controller.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>CallStateChanged</name>
        <value>ringon or ringoff</value>
    </api>
</envelope>
```

When the incoming call event is received, the LG Smart TV Host mutes the TV audio; and when it receives the call end event, the TV unmutes the TV audio.

[Table] Parameters of CallStateChanged

Parameter	Description
value	This value indicates an incoming call or the call end. The following two values are available. **ringon**: An incoming call is received **ringoff**: A call has ended

[Table] Actions of CallStateChanged

Item	Description
Persistent connection status	Since this event does not occur frequently, use Connection: Close.
Related commands	None

DragMode (Controller >> Host)

The Controller notifies that it will start or end drag & drop for an icon or an app on the Host screen.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>DragMode</name>
        <value>true or false</value>
    </api>
</envelope>
```

When the drag start event is received, the Host internally prepare to process drag & drop. Subsequently, when it receives the HandleTouchMove command from the Controller, the Host performs the drag process for the actual icons or apps on the screen. Lastly, when the drag end event is received, the Host drops the dragged object(s) in the current screen position of the Host. For details, see the usage scenario.

[Table] Parameters of DragMode

Parameter	Description
value	This value determines the start or end of the drag process. The following two values are available. **true**: Start dragging **false**: End dragging

[Table] Actions of DragMode

Item	Description
Persistent connection status	Since HandleTouchMove triggers after the dragging action, use Connection: Keep-Alive to request a persistent connection.
Related commands	HandleTouchMove command

3DMode (Controller << Host)

The 3DMode event notifies the Controller whether the LG Smart TV Host has entered or exited the 3D display mode.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>3DMode</name>
        <value>true or false</value>
    </api>
</envelope>
```

[Table] Parameters of 3DMode

Parameter	Description
value	**true**: Entered 3D mode **false**: Extied 3D mode

[Table] Actions of 3DMode

Item	Description
Persistent connection status	Since this event does not occur frequently, use Connection: Close to send the event to the Controller.
Related command and query	[Command] HandleKeyInput [Query] is_3d

Query (Controller >> Host)

Queries of the Remote Controller service complies with all common rules in the LG UDAP 2.0 Protocol Specification and with additional rules defined for queries. They use HTTP GET to fetch data from the Host.

http://target_ip:port/udap/api/data?target=apiName&paramName1=URL_Encode(param_value1)&paramName2=URL_Encode(param_value2)...

If a query is successfully performed, it returns HTTP/1.1 200 OK and the requested data. If there is an error, it returns an error code other than HTTP/1.1 200 OK without any data. For details of response codes, refer to the LG UDAP 2.0 Protocol Specification.

Queries are as follows:
- Current channel information (Controller >> Host)
- Entire channels list (Controller >> Host)
- Operation mode of the Host UI (Controller >> Host)
- Volume information of the Host (Controller >> Host)
- Obtaining the capture image of the Host (Controller >> Host)
- 3D mode of the Host (Controller >> Host)

Current channel information (Controller >> Host)

Fetches information on the currently broadcasting channel from the Host. Uses HTTP GET to fetch an XML document at the URL below.

http://target_ip:port/udap/api/data?target=cur_channel

The following XML document is returned when querying the current channel information.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <dataList name="Current Channel">
        <data>
            <chtype>channel type</chtype>
            <major>major number of the channel</major>
            <displayMajor>display major number of the channel</displayMajor>
            <minor>physical major number of the channel </minor>
            <displayMinor>minor number of the channel</displayMinor>
            <sourceIndex>source index of the channel</sourceIndex>
            <physicalNum>physical number of the channel</physicalNum>
            <chname>channel name</chname>
            <progName>current program name</progName>
            <audioCh>Audio channel or not</audioCh>
            <inputSourceName>fixed value as Antenna</inputSourceName>
            <inputSourceType>input source type</inputSourceType>
            <labelName>label name of the current source</labelName>
            <inputSourceIdx>input source index</inputSourceIdx>
        </data>
    </dataList>
</envelope>
```

[Table] Data Fields of Current Channel Information

Name of data field	Value of data field
chtype	Reception type of the currently broadcasting channel. The following values are available. **terrestrial**: Terrestrial broadcast **cable**: Cable broadcast **satellite**: Satellite broadcast
major	Indicates the physical major number of the current channel. For 11-1 MBC DTV, this value is 11.
displayMajor	Indicates the major number of the current channel. This value is used by the Controller UI.
minor	Indicates the physical minor number of the current channel. For 11-1 MBC DTV, this value is 1.
displayMinor	Indicates the minor number of the current channel. This value is used by the Controller UI.
sourceIndex	Source index of the current channel
physicalNum	Physical number of the current channel
chname	Indicates the name of the current channel. For 11-1 MBC DTV, this value MBC DTV.
progName	Name of the program currently being received on the current channel
audioCh	Shows whether the current channel is an audio channel. The following values are available. **0**: Normal channel **1**: Audio channel
inputSourceName	Name of the current input source. Fixed as **Antenna**.
inputSourceType	Type of the current input source. Fixed as 0.
labelName	Label name of the current input source
inputSourceIdx	Physical index value of the current input source

[Table] Actions of Current Channel Information

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related commands and events	None

Entire channels list (Controller >> Host)

Fetches information on the entire channels list from the Host. Uses HTTP GET to fetch an XML document at the URL below.

http://target_ip:port/udap/api/data?target=channel_list

The following XML document is returned when querying the entire channels list.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <dataList name="Channel List">
      <!-- Information of a channel -->
      <data>
         <chtype>channel type</chtype>
         <major>major number of the channel</major>
         <minor>minor number of the channel</minor>
         <sourceIndex>source index of the channel </sourceIndex>
         <physicalNum>physical major number of the channel </physicalNum>
         <chname>channel name</chname>
      </data>
         <!-- Channel information of different channels is listed -->
      <data>
         ...
      </data>
         ...
   </dataList>
</envelope>
```

[Table] Data Fields of Entire Channels List

Name of data field	Value of data field
chtype	Type of the current channel. The following values are available. **terrestrial**: Terrestrial broadcast **cable**: Cable broadcast **satellite**: Satellite broadcast
major	Indicates the major number of the current channel. For 11-1 MBC DTV, this value is 11.
minor	Indicates the minor number of the current channel. For 11-1 MBC DTV, this value is 1.
sourceIndex	Source index of the current channel
physicalNum	Physical number of the current channel
chname	Indicates the name of the current channel. For 11-1 MBC DTV, this value MBC DTV.

[Table] Actions of Current Channel Information

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related commands and events	None

Note

For the entire channels list, dataList is not divided into multiple dataLists according to the broadcast reception type (terrestrial, cable and satellite), but transferred as a single dataList (channel list) for compatibility with older versions of apps.

Operation mode of the Host UI (Controller >> Host)

Fetches information on the operation mode of the current UI from the Host. Uses HTTP GET to fetch an XML document at the URL below.

http://target_ip:port/udap/api/data?target=context_ui

The following XML document is returned when querying the entire channels list.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <dataList name="UI Mode">
        <data>
            <mode>UI mode value</mode>
        </data>
    </dataList>
</envelope>
```

[Table] Data Fields of Host UI Mode Information

Name of data field	Value of data field
mode	Indicates which one is better for easy control of the current UI mode of the Host between the magic remote Controller or the normal remote control. **VolCh**: Normal broadcast screen. It is better controlled by the channel up/down and volume up/down functions of the normal remote Controller. **TouchPad**: An app or the web browser is running on the Host. It is easier to use the mouse function of the magic remote Controller. See HandleTouchMove, HandleTouchClick, HandleTouchWheel and DragMode.

[Table] Action of the current channel info

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related commands and events	[Command] HandleKeyInput, HandleChannelChange, HandleTouchMove, HandleTouchClick, and HandleTouchWheel. [Event] DragMode

Volume information of the Host (Controller >> Host)

Fetches volume information from the Host.

http://target_ip:port/udap/api/data?target=volume_info

The following XML document is returned when querying the volume information.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <dataList name="Volume Info">
        <data>
            <mute>true or false</mute>
            <minLevel>Minimum volume level</minLevel>
            <maxLevel>Maximum volume level</maxLevel>
            <level>Current volume level</level>
        </data>
    </dataList>
</envelope>
```

[Table] Data Fields of Volume Information

Name of data field	Value of data field

Name of data field	Value of data field
mute	Indicates the mute status. The following two values are available. **true**: Audio muted **false**: Audio not muted
minLevel	Indicates the minimum volume level available for the Host. The value is 0.
maxLevel	Indicates the minimum volume level available for the Host. The usual value is 100.
level	Indicates the current volume level of the Host, in the range from minLevel to maxLevel.

[Table] Actions of Volume Information

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related command	HandleKeyInput

Obtaining the capture image of the Host (Controller >> Host)

Obtains the screen capture image of the Host.

http://target_ip:port/udap/api/data?target=screen_image

When the capture image is requested, the screen is captured as a 960x540 size of JPEG image and delivered to the Controller.

[Table] Actions for Obtaining Capture Image

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related commands and events	None

3D mode of the Host (Controller >> Host)

Obtains the current 3D mode from the Host.

http://target_ip:port/udap/api/data?target=is_3d

The following XML document is returned when querying the 3D mode of the Host.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <dataList name="is3D">
      <data>
         <is3D>true or false</is3D>
      </data>
   </dataList>
</envelope>
```

[Table] Data Fields of 3D mode information

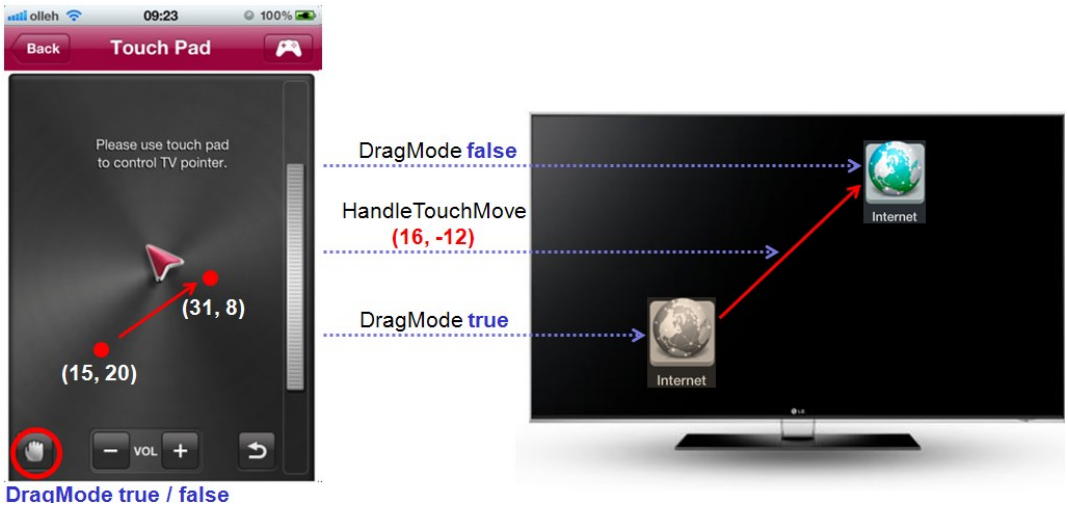
Name of data field	Valud of data field
is3D	Shows the 3D mode of the current Host. The following values are available. **true**: 3D mode **false**: Not 3D mode

[Table] Actions for 3D mode

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related event	3DMode

Usage Scenario

This example shows the actual scenario for the drag & drop action. When the drag start event is received from the Controller, the Host internally prepare to process drag & drop. Subsequently, when it receives the HandleTouchMove command from the Controller, the Host performs the drag process for the actual icons or apps on the screen. Lastly, when the drag end event is received, the Host drops the dragged object(s) in the current screen position of the Host. The diagram below shows an overview of this operation.



[Figure] Drag & Drop example

Text Input Service (smartText)

This chapter describes on Text Input service that allows the Controller to send text input to the Host or notify that text input is available.

This chapter includes the following sections.
- Discovery & Description
- Event (Controller <<->> Host)
- Usage Scenario

Discovery & Description

The ST value for searching the Text Input service is as follows.

```
urn:schemas-udap:service:smartText:1
```

If searching the SSDP with the ST value above, the descriptions are returned as below. The device section of the description XML contains the model name and the UUID value of the actual Host.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<envelope>
  <device>
    <deviceType>TV</deviceType>
    <modelName>LGSmartTV_33</modelName>
    <friendlyName>LG Smart TV</friendlyName>
    <manufacturer>LG Electronics</manufacturer>
    <uuid>0000004c-0600-4c04-0e04-0290040e900e</uuid>
  </device>
  <port>8080</port>
  <serviceList>
    <service name="smartText">
      <apiList>
        <api type="event">
          <name>TextEdited</name>
          <name>KeyboardVisible</name>
        </api>
      </apiList>
    </service>
  </serviceList>
</envelope>
```

Note

Since **udap:rootservice** searchs all services existing on Host, it can also be used by Text Input service and descriptions in Descriptions of All Service Profiles can be obtained.

Event (Controller <<->> Host)

Events of the Text Input service complies with all common rules in the LG UDAP 2.0 Protocol Specification and are sent in the HTTP POST method using the path below.

http://target_ip:port/udap/api/event

The XML format used for the event is the common format defined in the LG UDAP 2.0 Protocol Specification. Black fonts indicate fixed values, and red italic fonts indicate values variable depending on events.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>apiName</name>
        <paramName>param value</paramName>
        <!-- Additional parameters are listed if available -->
    </api>
</envelope>
```

Upon successful handling of an event, HTTP/1.1 200 OK is returned. For details of response codes, refer to the LG UDAP 2.0 Protocol Specification.

Events are as follows:
- KeyboardVisible (Controller << Host)
- TextEdited (Controller <<->> Host)

KeyboardVisible (Controller << Host)

If the Host is ready to receive text input from the Controller, the Host sends an event to the Controller, requesting to display the keyboard for text input. Also, while the Controller is entering text, if the Host becomes unavailable to receive the text input, the Host sends an event to the Controller, requesting to hide the keyboard.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>KeyboardVisible</name>
        <value>true or false</value>
        <mode>default or password</mode>
    </api>
</envelope>
```

When the Controller receives the event, it must take appropriate actions to show or hide the keyboard.

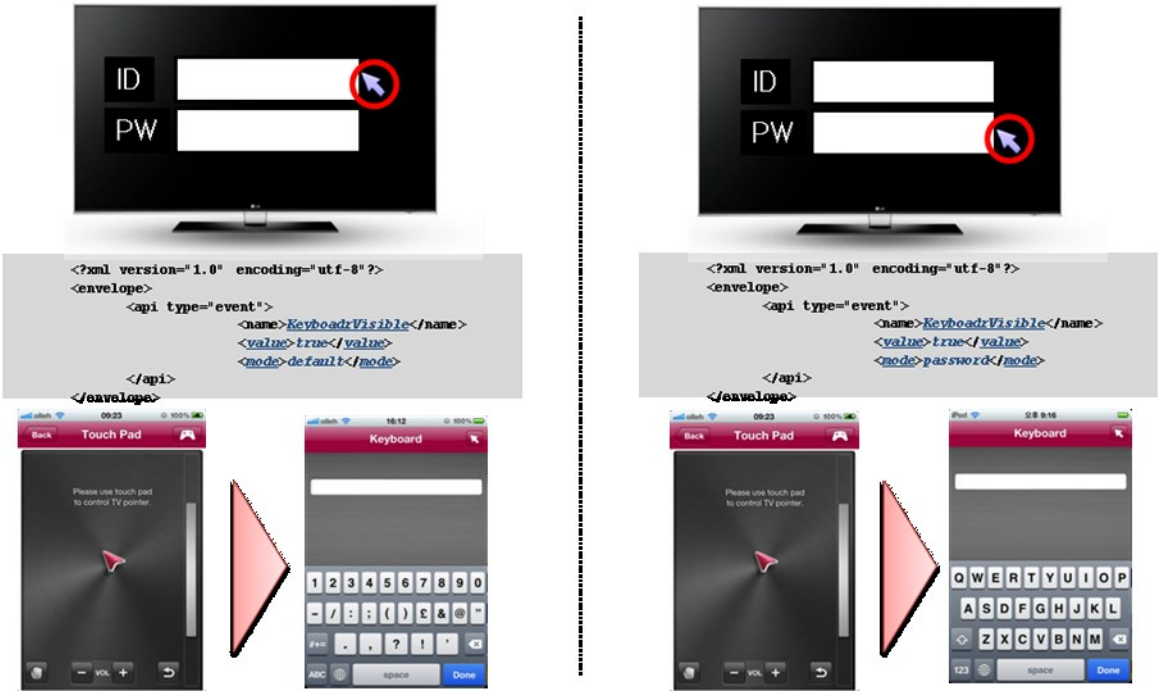
[Table] Parameters of KeyboardVisible

Parameters	Description
value	Indicates whether to show or hide the keyboard. The following two values are available. **true**: Show keyboard **false**: Hide keyboard
mode	Indicates the mode for showing the keyboard. The following two values are available. **default**: Normal keyboard mode. Shows the system default keyboard on the screen. **password**: Password input keyboard mode.

[Table] Action of KeyboardVisible

Actions	Description
Persistent connection status	Since this event does not occur frequently, use Connection: Close to send the event to the Controller.
Related events	HandleTouchClick event of the Remote Controller Service When a text input box of the Host is clicked, the KeyboardVisible event is triggered. TextEdited event of the Text Input service After sending the KeyboardVisible event to the Controller, the Host sends the TextEdited event.

Figure below provides a diagrammatic representation of the Controller action for the KeyboardVisible event.



[Figure] Processing of KeyboardVisible Event

TextEdited (Controller <<->> Host)

After sending the KeyboardVisible event to the Controller, the Host sends the entire string currently entered in the text input box of the Host to the Controller. The Controller must allow to insert the entire string received from the TextEdited event in the keyboard input window so that the current input string can be re-edited and sent. The Controller uses the same TextEdited event to send the string, that it was input, to the Host.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <api type="event">
      <name>TextEdited</name>
      <state>EditStart or Editing or EditEnd</state>
      <value> Entire string in the text input window </value>
   </api>
</envelope>
```

[Table] Parameters of TextEdited

Parameter	Description
state	This parameter indicates the current input state of the text input box. The following three values are available. **EditStart**: Start editing text **Editing**: Text editing in progress **EditEnd**: End editing text
value	Entire string entered in the text input box The string must be encoded in UTF-8. Up to 2KB are transferred. If the length of the string exceeds 2 KB, the first 2 KB are shown in the text input box of the receiver's device.

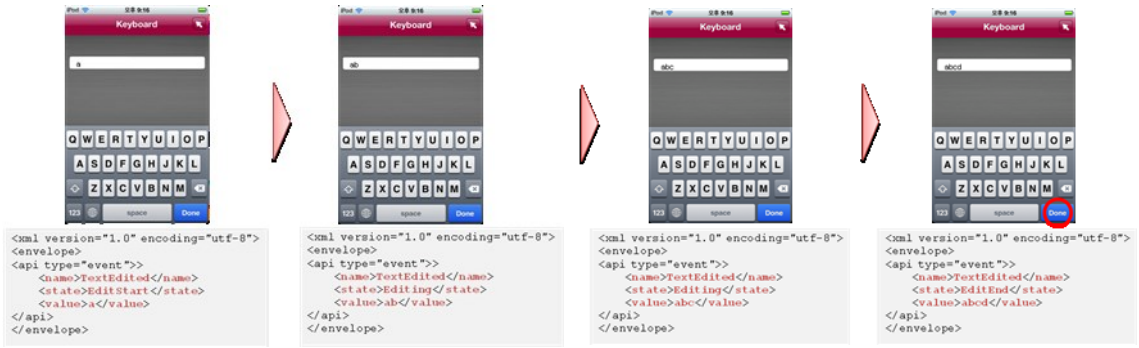
[Table] Actions of TextEdited

Item	Description
Persistent connection status	Since the Controller can quickly send the event according to the user's input speed, send the string over a persistent connection by using Connection: Keep-Alive.
Related events	KeyboardVisible

Figure below provides a diagrammatic representation of the process of sending TextEdited from the Host. The TextEdited event sent from the Host is sent only once immediately after the text input box on the Host is clicked and the KeyboardVisible event is generated. The input state is always set to Editing.

[Figure] Sending of TextEdited Event by Host

When the Controller sends the string entered in its text input box to the Host, it must send the input state of the text along with the entire string entered so far. For example, supposing that the text input screen is closed after entering ‘abcd’, the Controller sends the following message to the Host: (EditStart), ab (Editing), abc (Editing), abcd (EditEnd) Figure below provides a diagrammatic representation of this operation. If the TextEdited event received from the Host contains a string which is not null, the text input state starts editing.



[Figure] Sending of TextEdited Event by Controller

Usage Scenario

The figure below provides a diagrammatic representation of a usage scenario of the text input service. The following process is illustrated: The Host generates the KeyboardVisible event --> The Host sends the TextEdited event --> The Controller sends the TextEdited event.

[Figure] Usage Process of Text Input Service

Mobile Home Service (mobilehome)

This chapter describes on Mobile Home service that fetches the list of currently installed apps from the Host and

provides app functions, such as remote run and end, and other control functions.

This chapter includes the following sections.

Discovery & Description

The ST value for searching the Mobile Home service is as follows.

urn:schemas-udap:service:mobilehome:1

If searching the SSDP with the ST value above, the descriptions are returned as below. The device section of the description XML contains the model name and the UUID value of the actual Host.

```
<?xml version="1.0" encoding="utf-8"?>
▼<envelope>
  ▼<device>
      <deviceType>TV</deviceType>
      <modelName>LGSmartTV_33</modelName>
      <friendlyName>LG Smart TV</friendlyName>
      <manufacturer>LG Electronics</manufacturer>
      <uuid>0000004c-0600-4c04-0e04-0290040e900e</uuid>
    </device>
    <port>8080</port>
  ▼<serviceList>
    ▼<service name="mobilehome">
      ▼<apiList>
        ▼<api type="command">
            <name>AppTerminate</name>
            <name>AppExecute</name>
          </api>
        ▼<api type="event">
            <name>Mobilehome_App_Errstate</name>
            <name>Mobilehome_App_Change</name>
          </api>
        ▼<api type="query">
            <name>appicon_get</name>
            <name>appnum_get</name>
            <name>applist_get</name>
          </api>
        </apiList>
      </service>
    </serviceList>
</envelope>
```

Note

Since **udap:rootservice** searchs all services existing on Host, it can also be used by Mobile Home service and descriptions in Descriptions of All Service Profiles can be obtained.

Command (Controller >> Host)

Commands of the Mobile Home service complies with all common rules in the LG UDAP 2.0 Protocol Specification and are sent in the HTTP POST method using the path below.

http://target_ip:port/udap/api/command

The XML format used for the command is the common format defined in the LG UDAP 2.0 Protocol Specification. Black fonts indicate fixed values, and red italic fonts indicate values variable depending on commands.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="command">
        <name>apiName</name>
        <paramName>param value</paramName>
        <!-- Additional parameters are listed if available -->
    </api>
</envelope>
```

Upon successful handling of an command, HTTP/1.1 200 OK is returned. For details of response codes, refer to the LG UDAP 2.0 Protocol Specification.

Commands are as Follows:
- AppExecute (Controller >>Host)
- AppTerminate (Controller >> Host)

AppExecute (Controller >>Host)

This command starts a specific app on the Host. When the Host receives this command, it returns HTTP/1.1 200 OK regardless of the execution result. The actual execution result of the app can be determined by receiving the Mobilehome_App_Errstate event.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="command">
        <name>AppExecute</name>
        <auid>Unique ID of the app</auid>
        <appname>app name</appname>
        <contentId>Content ID</contentId>
    </api>
</envelope>
```

[Table] Parameters of AppExecute

Parameter	Description
auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.
appname	App name
contentId	This is the unique content ID assigned by the contents provider.

[Table] Actions of AppExecute

Item	Description
Persistent connection status	Since this command does not occur frequently, use Connection: Close to send the command to the Controller.
Related events and queries	[Event] Mobilehome_App_Errstate [Query] applist_get

AppTerminate (Controller >> Host)

This command terminates a specific app on the Host. When the Host receives this command, it returns HTTP/1.1 200 OK regardless of the execution result. The actual execution result of the app can be determined by receiving the Mobilehome_App_Errstate event.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="command">
        <name>AppTerminate</name>
        <auid>Unique ID of the app</auid>
        <appname>app name</appname>
    </api>
</envelope>
```

[Table] Parameters of AppTerminate

Parameters	Value
auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.
appname	App name

[Table] Actions of AppTerminate

Item	Description
Persistent connection status	Since this command does not occur frequently, use Connection: Close to send the command to the Controller.
Related events and queries	[Event] Mobilehome_App_Errstate [Query] applist_get

Event (Controller <<->> Host)

Events of the Mobile Home service complies with all common rules in the LG UDAP 2.0 Protocol Specification and are sent in the HTTP POST method using the path below.

http://target_ip:port/udap/api/event

The XML format used for the event is the common format defined in the LG UDAP 2.0 Protocol Specification. Black fonts indicate fixed values, and red italic fonts indicate values variable depending on events.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>apiName</name>
        <paramName>param value</paramName>
        <!– Additional parameters are listed if available -->
    </api>
</envelope>
```

Upon successful handling of an event, HTTP/1.1 200 OK is returned. For details of response codes, refer to the LG UDAP 2.0 Protocol Specification.

The events are as follows:
- Mobilehome_App_Errstate (Controller << Host)
- Mobilehome_App_Change (Controller << Host)

Mobilehome_App_Errstate (Controller << Host)

The Host sends the results for the AppExecute and AppTerminate commands, which are received from the Controller, to the Controller.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>Mobilehome App Errstate</name>
        <action>Execute or Terminate</action>
        <detail>Result of the action</detail>
    </api>
</envelope>
```

[Table] Parameters of Mobilehome_App_Errstate

Parameter	Description
action	This parameter is a value determining whether the processing result is for execution of the app or for termination of the app. The following two values are available. **Execute**: Indicates the result for execution of the app **Terminate**: Indicates the result for termination of the app
detail	Indicates the actual result value for execution or termination of the app. The following three values are available. **OK**: Execution or termination of the app is successful **ERROR**: Execution or termination of the app failed **BUSY**: The app to be executed is already running or the app to be terminated failed to be terminated due to a system problem of the Host

[Table] Actions of Mobilehome_App_Errstate

Item	Description
Persistent connection status	Since this event does not occur frequently, use Connection: Close to send the event to the Controller.
Related command	AppExecute, AppTerminate

Mobilehome_App_Change (Controller << Host)

The Mobilehome_App_Change event is sent to the Controller when an app is added to or removed from the Host.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
    <api type="event">
        <name>Mobilehome App Change</name>
        <action>Add or remove</action>
        <apptype>app type</apptype>
        <auid>Unique ID of the app</auid>
        <appname>app name</appname>
    </api>
</envelope>
```

[Table] Parameters of Mobilehome_App_Change

Parameter	Description
action	This parameter is a value determining whether the event is generated because an app is added to the Host or because an app is removed. The following 2 values are available. **Add**: An app is added **Delete**: An app is removed
apptype	This parameter is a value determining whether the app added or removed belongs to the Premium category or the My Apps category of the Host. The following two values are available. **2**: The app in the Premium category is added or removed **3**: The app in the My Apps category is added or removed
auid	This is the unique ID of the added or removed app, expressed as an 8-byte-long hexadecimal string.
appname	Name of the added or removed app

[Table] Actions of Mobilehome_App_Change

Item	Description
Persistent connection status	Since this event does not occur frequently, use Connection: Close to send the event to the Controller.
Related commands	AppExecute, AppTerminate

Query (Controller >> Host)

Queries of the Mobile Home service complies with all common rules in the LG UDAP 2.0 Protocol Specification and with additional rules defined for queries. They use HTTP GET to fetch data from the Host.

http://target_ip:port/udap/api/data?target=apiName&paramName1=URL_Encode(param_value1)&paramName2=URL_Encode(param_value2)...

If a query is successfully performed, it returns HTTP/1.1 200 OK and the requested data. If there is an error, it returns an error code other than HTTP/1.1 200 OK without any data. For details of response codes, refer to LG UDAP 2.0 Protocol Specification.

Queries are as follows:
- Obtaining the Apps list (Controller >> Host)
- Obtaining the number of Apps (Controller >> Host)
- Obtaining the icon image of App (Controller >> Host)

Obtaining the Apps list (Controller >> Host)

This is used for fetching the list of apps installed on the Host from the Premium menu and the My Apps menu on the Host.

http://target_ip:port/udap/api/data?target=applist_get&type=integer_value_type&index=integer_value_index&number=integer_value_number

The following XML document is returned when querying the app list.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <dataList name="App List">
      <data>
         <auid>Unique ID of the app</auid>
         <name>app name</name>
         <type>category of the app</type>
         <cpid>content ID</cpid>
         <adult>whether the app is adult all or not</adult>
         <icon_name> app icon name</icon_name>
      </data>
         <!-- Information of different apps are listed -->
      <data>
         ...
      </data>
          ...
   </dataList>
</envelope>
```

[Table] Parameters for Obtaining Apps List

Parameter	Value
type	This parameter specifies the category for obtaining the list of apps. The following three values are available. **1**: List of all apps **2**: List of apps in the Premium category **3**: List of apps in the My Apps category

Parameter	Value
index	Starting index of the apps list. The value range is from 1 to 1024.
number	This parameter specifies the number of apps to be obtained from the starting index. This value has to be greater than or equal to the index value. The value can be from 1 to 1024.

Note

If both index and number are 0, the list of all apps in the category specified by type is fetched.

[Table] Data field of obtaining Apps list

Data field name	Value
auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.
name	App name
type	Shows whether the app is under the Premium or My Apps category. The following values are available. **2**: App is under Premium category **3**: App is under My Apps category
cpid	Unique content ID given by Contents Provider
adult	Shows whether the app is an adult app. The following values are available. **true**: Adult app **false**: Not adult app
icon_name	App icon name

[Table] Actions of fetching Apps list

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related commands	AppExecute, AppTerminate

Note

For compatibility with older versions of apps, dataList is not divided into multiple dataLists according to the apps category type (Premium, My Apps), but transferred as a single dataList (app list).

Obtaining the number of Apps (Controller >> Host)

Obtains the number of apps in the Premium menu and the My Apps menu on the Host.

http://target_ip:port/udap/api/data?target=appnum_get&type=integer_value_type

Obtains the icon image of a specific app on the Host.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
   <dataList name="App List">
      <data>
         <type>category of the app</type>
         <number>number of the total apps under the category</number>
      </data>
   </dataList>
</envelope>
```

[Table] Parameters of obtaining the number of Apps

Parameter	Value
type	This parameter specifies the category for obtaining the list of apps. The following three values are available. **1**: List of all apps **2**: List of apps in the Premium category **3**: List of apps in the My Apps category

[Table] Data field of obtaining the number of Apps

Data field name	Value
type	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.
number	This parameter specifies the number of apps in the specified category. This value has to be greater than or equal to the index value. The value can be from 1 to 2048..

[Table] Actions of obtaining the number of App

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close.
Related command	AppExecute, AppTerminate

Obtaining the icon image of App (Controller >> Host)

Obtains the icon image of a specific app on the Host.

http://target_ip:port/udap/api/data?target=appicon_get&auid=hex_string_auid&appname=URL_Encode(string_appname)

When obtaining app icon is queried, the PNG type image data is sent to the Controller.

[Table] Data field of obtaining the icon image of App

Parameter	Value
Auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.
App name	App name

[Table] Actions of obtaining the icon image of App

Item	Description
Persistent connection status	Since as many queries as desired can be requested continuously using the auid and appname values returned from the apps list, request over a persistent connection by using Connection: Keep-Alive.
Related query	applist_get

Usage Scenario

The figure below shows the process of obtaining the apps list, fetching and displaying the icon images of apps and executing a specific app on the Host when the app icon is touched.

[Figure] Using Mobile Home Service

App To App Service (AppToApp)

This chapter provides definitions of general methods in which the apps on the Host and the Controller communicate and interact. While the Mobile Home service includes definitions of some communications of apps between the Host and the Controller, this chapter provides definitions of more general methods.

This chapter includes the following sections.

Discovery & Description

The ST value for searching the App to App service is as follows.

urn:schemas-udap:service:AppToApp:1

If searching the SSDP with the ST value above, the descriptions are returned as below. The device section of the description XML contains the model name and the UUID value of the actual Host.

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
  <device>
    <deviceType>TV</deviceType>
    <modelName>LGSmartTV_33</modelName>
    <friendlyName>LG Smart TV</friendlyName>
    <manufacturer>LG Electronics</manufacturer>
    <uuid>0000004c-0600-4c04-0e04-0290040e900e</uuid>
  </device>
  <port>8080</port>
  <serviceList>
    <service name="AppToApp">
      <apiList>
        <api type="command">
          <name>SendMessage</name>
          <name>TerminateApplication</name>
          <name>LaunchApplication</name>
        </api>
        <api type="event">
          <name>ReceiveMessage</name>
        </api>
        <api type="query">
          <name>GetApplicationStatus</name>
          <name>GetApplicationID</name>
        </api>
      </apiList>
    </service>
  </serviceList>
</envelope>
```

Note

Since **udap:rootservice** searches all services existing on Host, it can also be used by App to App service and descriptions in **Descriptions of All Service Profiles** can be obtained.

Command (Controller >> Host)

Command of the App to App service complies with all **common rules** in the LG UDAP 2.0 Protocol Specification and are sent in the HTTP POST method using the path below.

http://target_ip:port/udap/api/apptoapp/command/...

Content format used in the command sends different values through each path, which differs from the common formats defined in the **LG UDAP 2.0 Protocol Specification**.

Upon successful handling of an command, HTTP/1.1 200 OK is returned. For details of response codes, refer to the **LG UDAP 2.0 Protocol Specification**.

Commands are as follows:
- **Launch Application (Controller >> Host)**
- **Terminate Application (Controller >> Host)**
- **Send Message (Controller >> Host)**

Launch Application (Controller >> Host)

This command starts a specific app on the Host. When the Host receives this command, it uses the return value of the command to determine the success/failure status. (For example, HTTP/1.1 200 OK)

```
http://target_ip:port/udap/api/apptoapp/command/{AUID}/run
```

[Table] Parameters of App Execute

Parameter	Description
auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.

[Table] Actions of Launch App

Item	Description
Persistent connection status	Since this command does not occur frequently, use Connection: Close to send the command to the Controller.
Related events and queries	Get Application AUID

[Table] Return values of Launch App (only different parts from the spec in meaning are described)

Return value	Description
400	Bad Request : Wrong format requested
503	Service Unavailable : There is no corresponding app for the AUID

Terminate Application (Controller >> Host)

This command terminates an specific app on the Host. When the Host receives this command, it determines the success/failure status with a returned value. (For example, HTTP/1.1 200 OK)

```
http://target_ip:port/udap/api/apptoapp/command/{AUID}/term
```

[Table] Parameters of Terminate App

Parameter	Description
auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.

[Table] Actions of Terminate App

Item	Description
Persistent connection status	Since this command does not occur frequently, use Connection: Close to send the command to the Controller.
Related events and queries	Get Application AUID

Send Message (Controller >> Host)

This command sends messages to a specific web app on the Host. When the Host receives this command, it determines the success/failure status with a returned value. (For example, HTTP/1.1 200 OK) Only an web app can receive this value. For the JavaScript API of the web app, see the “AppToApp Plugin and API” in **Developing > API** section in this Library.

```
http://target_ip:port/udap/api/apptoapp/command/{AUID}/send
```

[Table] Parameters of Send Message

Parameter	Description

Parameter	Description
auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.

[Table] Actions of Send Message

Item	Description
Persistent connection status	If sending a single message to the Host, use Connection: Close. To send multiple messages continuously and quickly to the Host, enable the persistent connection by using Connection: Keep-Alive. For detailed description on Persistent Connection, see LG UDAP 2.0 Protocol Specification.
Related events and queries	Receive Message event

Event (Controller <<->> Host)

Events of the App to App service complies with all common rules in the LG UDAP 2.0 Protocol Specification and are sent in the HTTP POST method using the path below.

http://target_ip:port/udap/api/apptoapp/event/...

Upon successful handling of an event, HTTP/1.1 200 OK is returned. For details of response codes, refer to the LG UDAP 2.0 Protocol Specification.

Event is as follows:
• Receive Message (Controller << Host)

Receive Message (Controller << Host)

This uses a JavaScript code of a web app on the Host to send messages to the Controller. At this point, the message is the content sent from the TV app. For information on how to send from web apps, see the “AppToApp Plugin and API” in **Developing > API** section in this Library. (the AUID sent from the Host to Controller is currently not used. The value is always set to “0”.)

```
http://target_ip:port/udap/api/apptoapp/event/{AUID}/send
```

```
POST /udap/api/apptoapp/event/0/send HTTP/1.1
Host: 192.168.0.150:8080
Content-Type: text/plain; charset=utf-8
Content-Length: 44
Connection: Close
User-Agent: Linux/3.0.13 UDAP/2.0 GLOBAL-PLAT4/2.0
TV App Message!
```

[Table] Actions of Receive Message

Item	Description
Persistent connection status	Since this event assumes that multiple messages are continuously and quickly sent to the Host, enable the persistent connection by using Connection: Keep-Alive. For detailed description on Persistent Connection, see LG UDAP 2.0 Protocol Specification.
Related command	Send Message command

Query (Controller >> Host)

Queries of the App to App service complies with all common rules in the LG UDAP 2.0 Protocol Specification and with additional rules defined for queries. They use HTTP GET to fetch data from the Host.

http://target_ip:port/udap/api/apptoapp/data/....

If a query is successfully performed, it returns HTTP/1.1 200 OK and the requested data. If there is an error, it returns an error code other than HTTP/1.1 200 OK without any data. For details of response codes, refer to **LG UDAP 2.0 Protocol Specification**.

Queries are as follows:
- **Get Application AUID (Controller >> Host)**
- **Get Application Status (Controller >> Host)**

Get Application AUID (Controller >> Host)

This is used for obtaining the auid of an installed app by using the app name. For example, the app name “My First Web App” can be used for obtaining the auid.

http://target_ip:port/udap/api/apptoapp/data/{Application Name}

The example below is for obtaining the auid of an app named “abc”. In this case, the app name “abc” must be URL encoded.

```
GET /udap/api/apptoapp/data/abc HTTP/1.1
User-Agent: UDAP/2.0
Connection: Close
Host: 192.168.0.153:8080
```

The following text document is returned when querying for the auid of an app. The content is the auid value.

```
HTTP/1.1 200 OK
SERVER: Linux/3.0.13 UDAP/2.0 GLOBAL-PLAT4/2.0
Cache-Control: no-store, no-cache, must-revalidate
Date: Fri Sep 21 06:31:40 2012 GMT
Connection: Close
Content-Type: text/plain; charset=utf-8
Content-Length: 5
12345
```

[Table] Parameters of Get Application AUID

Parameter	Value
Application Name	This indicates the name of the app (app title used for app registration). It must be URL encoded. (e.g. “ABC DEF” → “ABC%20DEF”)

[Table] Data field of Get Application AUID

Data field name	Value
N/A	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.

[Table] Actions of Get Application AUID

Item	Description
Persistent connection status	Since this query does not occur frequently, use Connection: Close..
Related command	**Launch App**, **Terminate App**

Note

Apps are searched in the Premium, My Apps, USB and Downloaded categories in order. If there are duplicate app names, the first auid found is returned. For example, if there are apps with the same name in Premium and USB, the auid of the app in Premium is returned.

Get Application Status (Controller >> Host)

This is used for obtaining the current state of an app on the Host. This shows whether the app is loading, terminating or running.

http://target_ip:port/udap/api/apptoapp/data/{AUID}/status

```
GET /udap/api/apptoapp/data/12345/status HTTP/1.1
User-Agent: UDAP/2.0
Host: 192.168.0.153:8080
Connection: Keep-Alive
```

The following XML document is returned when querying the app status.

```
SERVER: Linux/3.0.13 UDAP/2.0 GLOBAL-PLAT4/2.0
Cache-Control: no-store, no-cache, must-revalidate
Date: Fri Sep 21 06:03:39 2012 GMT
Connection: Keep-Alive
Keep-Alive: timeout=12, max=172800
Content-Type: text/plain; charset=utf-8
Content-Length: 4
NONE
```

[Table] Data Fields of Obtaining App State

Data filed name	Value
auid	This is the unique ID of the app, expressed as an 8-byte-long hexadecimal string.

[Table] Data field of obtaining App status

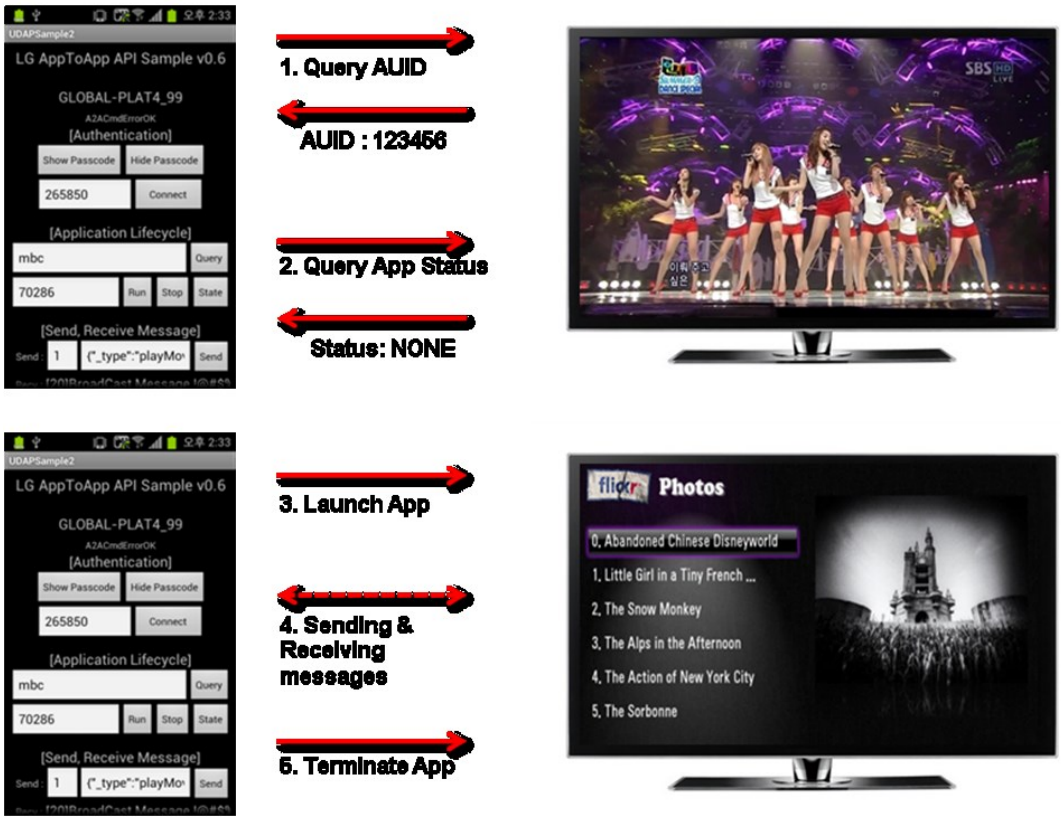
Data filed name	Value
N/A	Available values are NONE (inactive), LOAD (app is loading), RUN (app is running and on focus), RUN_NF (app is running and off focus) and TERM (app is terminating)

[Table] Actions of obtaining App status

Item	Description
Persistent connection status	To query the Host for app state continuously and quickly, enable the persistent connection by using Connection: Keep-Alive. For detailed description on Persistent Connection, see LG UDAP 2.0 Protocol Specification.
Related command	Launch App, Terminate App

Usage Scenario

The figure below illustrates the process and concept of communications between the Host and the Controller using the App To App service.



[Figure] Basic AppToApp Workflow

Annex A Table of virtual key codes on remote Controller

The table below shows virtual key codes of the remote Controller keys used by the HandleKeyInput command in **Command**. To send virtual key codes to the Host using HandleKeyInput, assign appropriate values for desired purposes by referring to the table below.

[Table] Virtual key codes on remote Controller

Virtual key code (decimal number)	Description
1	POWER
2	Number 0
3	Number 1
4	Number 2
5	Number 3
6	Number 4
7	Number 5
8	Number 6
9	Number 7
10	Number 8
11	Number 9

Virtual key code (decimal number)	Description
12	UP key among remote Controller’s 4 direction keys
13	DOWN key among remote Controller’s 4 direction keys
14	LEFT key among remote Controller’s 4 direction keys
15	RIGHT key among remote Controller’s 4 direction keys
20	OK
21	Home menu
22	Menu key (same with Home menu key)
23	Previous key (Back)
24	Volume up
25	Volume down
26	Mute (toggle)
27	Channel UP (+)
28	Channel DOWN (-)
29	Blue key of data broadcast
30	Green key of data broadcast
31	Red key of data broadcast
32	Yellow key of data broadcast
33	Play
34	Pause
35	Stop
36	Fast forward (FF)
37	Rewind (REW)
38	Skip Forward
39	Skip Backward
40	Record
41	Recording list
42	Repeat
43	Live TV
44	EPG
45	Current program information
46	Aspect ratio
47	External input

Virtual key code (decimal number)	Description
48	PIP secondary video
49	Show / Change subtitle
50	Program list
51	Tele Text
52	Mark
400	3D Video
401	3D L/R
402	Dash (-)
403	Previous channel (Flash back)
404	Favorite channel
405	Quick menu
406	Text Option
407	Audio Description
408	NetCast key (same with Home menu)
409	Energy saving
410	A/V mode
411	SIMPLINK
412	Exit
413	Reservation programs list
414	PIP channel UP
415	PIP channel DOWN
416	Switching between primary/secondary video
417	My Apps