Cloud Transport: 1 Service

For UPnP Version 2.0

Status: Standardized DCP (SDCP)

Date: December 31, 2015

Document Version: 1.0

Service Template Version: 2.0.0

THIS APPROVED SPECIFICATION WAS COMPLETED PRIOR TO THE COMBINATION OF UPNP INTO THE OPEN CONNECTIVITY FOUNDATION. ALL LICENSES, INTELLECTUAL PROPERTY RIGHTS, AND OTHER RIGHTS, RESPONSIBILITIES, OBLIGATIONS, STANDARDS, AND PROTOCOLS ASSOCIATED WITH THIS APPROVED SPECIFICATION ARE SUBJECT TO THE UPNP BYLAWS AND FORUM MEMBERSHIP AGREEMENT.

Legal Disclaimer

NOTHING CONTAINED IN THIS DOCUMENT SHALL BE DEEMED AS GRANTING YOU ANY KIND OF LICENSE IN ITS CONTENT, EITHER EXPRESSLY OR IMPLIEDLY, OR TO ANY INTELLECTUAL PROPERTY OWNED OR CONTROLLED BY ANY OF THE AUTHORS OR DEVELOPERS OF THIS DOCUMENT. THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "AS IS" BASIS, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE AUTHORS AND DEVELOPERS OF THIS SPECIFICATION HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, STATUTORY OR AT COMMON LAW, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OPEN CONNECTIVITY FOUNDATION, INC. FURTHER DISCLAIMS ANY AND ALL WARRANTIES OF NON-INFRINGEMENT, ACCURACY OR LACK OF VIRUSES.

The OCF logo is a trademark of Open Connectivity Foundation, Inc. in the United States or other countries. *Other names and brands may be claimed as the property of others.

Copyright © 2015 Open Connectivity Foundation, Inc. All rights reserved.

Copying or other form of reproduction and/or distribution of these works are strictly prohibited

<table>
<thead>
<tr>
<th>Authors a</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clarke Stevens</td>
<td>Cablelabs</td>
</tr>
<tr>
<td>Wouter van der Beek (Chair)</td>
<td>Cisco</td>
</tr>
<tr>
<td>Bich Nguyen</td>
<td>GoPro</td>
</tr>
<tr>
<td>Keith Miller</td>
<td>BKM Systems Group, LLC</td>
</tr>
</tbody>
</table>

a The UPnP forum in no way guarantees the accuracy or completeness of this author list and in no way implies any rights for or support from those members listed. This list is not the specifications’ contributor list that is kept on the UPnP Forum’s website.
# CONTENTS

1 Scope ............................................................................................................................... 4  
  1.1 Introduction ............................................................................................................. 4  
2 Normative References .................................................................................................. 4  
3 Terms, Definitions and Abbreviations ........................................................................... 5  
  3.1 Non-Restrictable ..................................................................................................... 5  
  3.2 Restrictable ............................................................................................................. 5  
4 Security Feature ............................................................................................................. 5  
  4.1 Device Protection .................................................................................................... 5  
  4.2 Restrictable and Non-Restrictable actions ............................................................... 6  
5 Notations and conventions ............................................................................................... 6  
  5.1 Notation .................................................................................................................. 6  
  5.2 Data Types .............................................................................................................. 7  
  5.3 Vendor-defined Extensions ...................................................................................... 7  
6 Service Modelling Definitions ........................................................................................ 7  
  6.1 Service Type .......................................................................................................... 7  
  6.2 CloudTransport:1 Service Architecture ................................................................... 7  
  6.3 Key Concepts .......................................................................................................... 8  
  6.4 State Variables ....................................................................................................... 11  
    6.4.1 State Variable Overview ............................................................................ 11  
    6.4.2 State Variable A_ARG_TYPE_Host ........................................................... 11  
    6.4.3 State Variable A_ARG_TYPE_Method ...................................................... 11  
    6.4.4 State Variable A_ARG_TYPE_Identifier .................................................. 11  
    6.4.5 State Variable A_ARG_TYPE_Headers ..................................................... 12  
    6.4.6 State Variable A_ARG_TYPE_Body ........................................................... 12  
    6.4.7 State Variable A_ARG_TYPE_UI4 .............................................................. 12  
    6.4.8 State Variable A_ARG_TYPE_Flag ............................................................ 12  
  6.5 Eventing and Moderation ....................................................................................... 12  
  6.6 Actions .................................................................................................................. 12  
    6.6.1 Introduction ............................................................................................... 12  
    6.6.2 HTTPConnectMethod() ........................................................................... 13  
    6.6.3 HTTPWriteHeaders() ................................................................................. 15  
    6.6.4 HTTPReadHeaders() ................................................................................... 16  
    6.6.5 HTTPReadBody() .......................................................................................... 18  
    6.6.6 HTTPWriteBody() ........................................................................................ 20  
7 Theory of Operations (informative) ................................................................................. 22  
  7.1 General Usage Scenarios ...................................................................................... 22  
    7.1.1 HTTP HEAD REQUEST and RESPONSE .................................................. 22  
    7.1.2 HTTP GET REQUEST and RESPONSE .................................................... 25  
    7.1.3 HTTP GET REQUEST with "Chunked” RESPONSE ................................... 29  
    7.1.4 HTTP POST REQUEST with "100-Continue” ............................................ 30  
    7.1.5 Closing an HTTP Connection ..................................................................... 30  
8 XML Service Description ............................................................................................. 30
Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles ........................................6
Table 6-1 — State Variables .....................................................................................................11
Table 6-2 — Eventing and Moderation ..................................................................................12
Table 6-3 — Actions ...............................................................................................................13
Table 6-4 — Typical Action Sequences for HTTP REQUEST and RESPONSE ..................13
Table 6-5 — Arguments for HTTPConnectMethod() ...........................................................14
Table 6-6 — Error Codes for HTTPConnectMethod() .......................................................15
Table 6-7 — Arguments for HTTPWriteMethod() ...............................................................15
Table 6-8 — Error Codes for HTTPWriteMethod() .............................................................16
Table 6-9 — Arguments for HTTPReadHeaders() ..............................................................17
Table 6-10 — Error Codes for HTTPReadHeaders() .........................................................18
Table 6-11 — Arguments for HTTPReadBody() ...............................................................19
Table 6-12 — Error Codes for HTTPReadBody() ..............................................................20
Table 6-13 — Arguments for HTTPWriteBody() ...............................................................21
Table 6-14 — Error Codes for HTTPWriteBody() ..............................................................22

Figure 6-1 CloudTransport Architecture UDA-to-UDA and UDA-to-UCA .........................8
Figure 6-2 CloudTransport Generalized Call Flow ..............................................................10
Figure 7-1 HTTP HEAD REQUEST and RESPONSE ..........................................................23
Figure 7-2 HTTP GET REQUEST and RESPONSE Part 1 ....................................................26
Figure 7-3 HTTP GET REQUEST and RESPONSE Part 2 ....................................................27
1 SCOPE

1.1 INTRODUCTION

This document defines the service CloudTransport:1, which identifies Version 1 of the service named CloudTransport:1. This Publicly Available Specification is applicable to Standardized DCPs of the UPnP Forum which include this service.

This service definition is compliant with the UPnP Device Architecture, version 2.0.

2 NORMATIVE REFERENCES

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.


3 TERMS, DEFINITIONS AND ABBREVIATIONS

For the purposes of this document, the terms and definitions given in [UDA], [CDS4], [CPROXY], [DP], and [PROXY] apply.

3.1 NON-RESTRICTABLE

A category of action that, when the DeviceProtection [DP] service is implemented on the device, cannot be blocked according to the presence or absence of a specific Role attached to a Control Point Identity or User Identity. See [CDS4] for further explanation of Role, Control Point Identity and User Identity.

3.2 RESTRICTABLE

A category of actions that, when the DeviceProtection [DP] service is implemented on the device, can be blocked according to the presence or absence of a specific Role attached to a Control Point Identity or User Identity.

4 SECURITY FEATURE

4.1 DEVICE PROTECTION

The CloudTransport Service should be implemented on a device supporting the DeviceProtection service [DP]. If the DeviceProtection service is implemented it shall support the following roles:

- **Public** – A control point with the Public role can successfully invoke any of the actions as long as the target connection is not an HTTPS session.
• **Basic** – A control point with the Basic role can successfully invoke any of the actions as it will have been authenticated over a TLS connection.

• **Admin** – A control point with the Admin role can successfully invoke any of the actions as it will have been authenticated over a TLS connection.

If the DeviceProtection service is not implemented then HTTPS connections are only supported for UCC-CPs actions on the UCA interface.

### 4.2 RESTRICTABLE AND NON-RESTRICTABLE ACTIONS

The CloudTransport service actions defined in this specification have the Restrictable, Non-Restrictable assignments as indicated in Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles.

#### Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Restrictable/Non-Restrictable to Indicated Role</th>
<th>No DeviceProtection</th>
<th>Public</th>
<th>Basic</th>
<th>Admin</th>
<th>UCC-CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPConnectMethod()</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td></td>
</tr>
<tr>
<td>HTTPReadHeaders()</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td></td>
</tr>
<tr>
<td>HTTPWriteHeaders()</td>
<td>HTTP</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td></td>
</tr>
<tr>
<td>HTTPRequestBody()</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td></td>
</tr>
<tr>
<td>HTTPResponseBody()</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td>HTTP, HTTPS</td>
<td></td>
</tr>
</tbody>
</table>

### 5 NOTATIONS AND CONVENTIONS

#### 5.1 NOTATION

- Strings that are to be taken literally are enclosed in “double quotes”.
- Words that are emphasized are printed in *italic*.
- Keywords that are defined by the UPnP Working Committee are printed using the *forum* character style.
- Keywords that are defined by the UPnP Device Architecture are printed using the *arch* character style.
- Keywords that are defined specific to the UPnP Device Architecture Annex C are printed using *UCA* character style.
- Keywords that are defined specific to XMPP are printed using *XMPP* character style.
- A double colon delimiter, “::”, signifies a hierarchical parent-child (parent::child) relationship between the two objects separated by the double colon. This delimiter is used in multiple contexts, for example: Service::Action(), Action::Argument, parentProperty::childProperty.

---

1. An HTTP value in the table indicates that the CloudTransport service is allowed to communicate using unencrypted HTTP REQUESTs and RESPONSEs on a local connection when the invoking control point either has the indicated Role, has no Role in the case of no DeviceProtection support or is a UCC-CP communicating on the UCA interface, that is HTTP communication is *Non-Restrictable*.

An HTTPS value in the table indicates that the CloudTransport service is allowed to communicate using encrypted HTTP REQUESTs and RESPONSEs on a local connection when the invoking control point either has the indicated Role or is an UCC-CP communicating on the UCA interface, otherwise a service specific 704 ErrorCode is issued, that is HTTPS communication is *Restrictable*.
5.2 DATA TYPES

This specification uses data type definitions from two different sources. The UPnP Device Architecture defined data types are used to define state variable and action argument data types UPnP Device Architecture, version 2.0 [UDA]. The XML Schema namespace is used to define property data types XML Schema Part 2: Data Types, Second Edition [XSD 2.0].

For UPnP Device Architecture defined Boolean data types, it is strongly recommended to use the value “0” for false, and the value “1” for true. The values “true”, “yes”, “false”, or “no” may also be used but are not recommended. The values “yes” and “no” are deprecated and shall not be sent out by devices but shall be accepted on input.

For XML Schema defined Boolean data types, it is strongly recommended to use the value “0” for false, and the value “1” for true. The values “true”, “yes”, “false”, or “no” may also be used but are not recommended. The values “yes” and “no” are deprecated and shall not be sent out by devices but shall be accepted on input.

5.3 VENDOR-DEFINED EXTENSIONS

Whenever vendors create additional vendor-defined state variables, actions or properties, their assigned names and XML representation shall follow the naming conventions and XML rules as specified in UPnP Device Architecture, version 2.0 [UDA], Clause 2.5, “Description: Non-standard vendor extensions”.

6 SERVICE MODELLING DEFINITIONS

6.1 SERVICE TYPE

The following URN identifies a service that is compliant with this specification:

urn:schemas-upnp-org:service:CloudTransport:1

CloudTransport service is used herein to refer to this service type.

6.2 CLOUDTRANSPORT:1 SERVICE ARCHITECTURE

The CloudTransport service consists of these architectural elements:

- A device hosting the CloudTransport service, this device may be a UDA (local) device or a UCCD.
- An HTTP Client for each connection (typically local) that the CloudTransport service will support that can interact with the CloudTransport service.
- Internal buffering for temporary storage of the HTTP REQUEST and RESPONSE messages on the device or UCCD implementing the CloudTransport service.

Two typical configurations are illustrated in Figure 6-2. The first being a scenario involving a CloudTransport service connecting two separate local networks the second being a scenario connecting a UCC-CP to a local network.
6.3 KEY CONCEPTS

The CloudTransport service provides the functionality to relay HTTP REQUESTs and RESPONSEs over a remote endpoint. It is suitable for usage over a UCA interface, as well as, a UDA (local) interface; although it could be used to connect UDA devices on the same local network, or different local networks, its main applicability will be for connecting UCA entities to UDA (local) resources. The UCA use cases are emphasized in this specification. For example, a content item available on a MediaServer with a resource of:

```
<res protocolInfo="http-get::*:audio/mpeg:*">
  http://192.168.0.1/audio/goodsong.mp3
</res>
```
would be reachable for devices and control points on the local network but not normally to UCCDs and UCC-CPs not connected to the same local network.

Utilizing the CloudTransport service, new actions are made available to CloudTransport control points, UCC-CPs or UCCDs (via an embedded UCC-CP) allowing them to open a local connection on the UDA side of a device (or UCCD), via an embedded HTTP Client, and execute HTTP REQUEST and RESPONSE transactions on the same or different UDA (local) interface or the UDA side of a UCCD. On the UCA side the, HTTP REQUESTs and RESPONSEs are distinguished by HEADER and BODY data type arguments, encoded and decoded using BASE64 transformations as needed for the CloudTransport actions and sent as SOAP or SOAP over XMPP messages. Also, because the UCA (XMPP) message sizes will typically be smaller than HTTP REQUEST over the local network some buffering will be needed on the device supporting the CloudTransport service, as well as, fragmentation and re-assemble of the HTTP stream. No lower level transport, that is TCP, occurs over the UCA interface.
Figure 6-2 CloudTransport Generalized Call Flow

The CloudTransport Service has both a UDA and UCA interface on its host UCCD, as well as a UDA side embedded CP and HTTP Client.

HTTP REQUEST and RESPONSE UCA and UDA typical exchange

The actual HTTP Request is then relayed to the Legacy Device via the embedded CP on the UDA Inf.

The actual HTTP Response is then returned to the embedded CP on the UDA Inf.

The UC-C-P sends an HTTP Request action to the CloudTransport UCA Inf for the target <res>.

That HTTP Response is then relayed back to UCC-CP over the UDA Inf.
6.4 STATE VARIABLES

Note: For first-time reader, it may be more insightful to read the theory of operations first and then the action definitions before reading the state variable definitions.

6.4.1 State Variable Overview

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>R/A</th>
<th>Data Type</th>
<th>Allowed Value</th>
<th>Default Value</th>
<th>Eng. Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ARG_TYPE_Host</td>
<td>R</td>
<td>String</td>
<td>See 6.4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Headers</td>
<td>R</td>
<td>string</td>
<td>See 6.4.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Identifier</td>
<td>R</td>
<td>string</td>
<td>See 6.4.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Headers</td>
<td>R</td>
<td>string</td>
<td>See 6.4.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Body</td>
<td>R</td>
<td>string</td>
<td>See 6.4.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_UI4</td>
<td>R</td>
<td>ui4</td>
<td>See 6.4.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Flag</td>
<td>R</td>
<td>boolean</td>
<td>See 6.4.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-standard state variables implemented by a UPnP vendor go here</td>
<td>X</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>

NOTES:

a For a device this column indicates whether the state variable shall be implemented or not, where R = required, A = allowed, CR = conditionally required, CA = conditionally allowed, X = Non-standard, add -D when deprecated (e.g., R-D, A-D).

b CSV stands for Comma-Separated Value list. The type between brackets denotes the UPnP data type used for the elements inside the list. The CSV list concept is defined more formally in the ContentDirectory service template.

c See referenced subclause for conditions under which the implementation of this state variable is required.

6.4.2 State Variable A_ARG_TYPE_Host

This required state variable provides type information for identifying an HTTP server endpoint on the local network that the CloudTransport service is to connect to. It is of the form "host[:port]" where the value of the host part is an IPv4 or IPv6 literal and optional [post] part is a string value corresponding to an unsigned 2 byte integer. An example would be "192.168.0.5:37222". The data type is string.

6.4.3 State Variable A_ARG_TYPE_Method

This required state variable provides type information for identifying a complete HTTP METHOD REQUEST line to be used by the CloudTransport service action to connect to a local device on a UDA. The data type is string.

6.4.4 State Variable A_ARG_TYPE_Identifier

This required state variable provides type information for identifying a specific HTTP connection in use by the CloudTransport service and tying the UCA side to a specific connection on the UDA side. The data type is string.
### 6.4.5 State Variable **A_ARG_TYPE_Hosts**

This required state variable provides type information for HTTP HEADER data for CloudTransport service actions. The data type is `string`.

### 6.4.6 State Variable **A_ARG_TYPE_Body**

This required state variable provides type information for HTTP Body data for CloudTransport service actions. The data type is `base64`.

### 6.4.7 State Variable **A_ARG_TYPE_UI4**

This required state variable provides type information for indicating the size (length) of operations of type read and write on HTTP REQUEST and RESPONSE for CloudTransport service actions. The data type is `ui4`.

### 6.4.8 State Variable **A_ARG_TYPE_Flag**

This required state variable provides type information for indicating a Boolean "1" or "0" ("true" or "false") input or output argument. The data type is `boolean`.

### 6.5 EVENTING AND MODERATION

The CloudTransport service has no evented state variables.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Evented</th>
<th>Moderation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Moderated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YES</td>
</tr>
</tbody>
</table>

### 6.6 ACTIONS

#### 6.6.1 Introduction

The CloudTransport service defines the actions in Table 6-3 and uses specific combinations to realize HTTP REQUEST and RESPONSE messaging across a UCA interface with an HTTP endpoint on a local network.
Table 6-3 — Actions

<table>
<thead>
<tr>
<th>Name</th>
<th>Device R/A</th>
<th>Control Point R/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPConnectMethod()</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>HTTPWriteHeaders()</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>HTTPReadHeaders()</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>HTTPReadBody()</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>HTTPWriteBody()</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

a For a device this column indicates whether the action shall be implemented or not, where R = required, O = allowed, CR = conditionally required, CA = conditionally allowed, X = Non-standard, add -D when deprecated (e.g., R-D, O-D).

b For a control point this column indicates whether a control point shall be capable of invoking this action, where R = required, A = allowed, CR = conditionally required, CA = conditionally allowed, X = Non-standard, add -D when deprecated (e.g., R-D, O-D).

By using the actions in a specific order, complete HTTP transactions such as HEAD, DELETE, GET, POST and PUT can be realized. Table 6-4 shows the typical action sequence for realizing an HTTP REQUEST and RESPONSE transaction.

Table 6-4 — Typical Action Sequences for HTTP REQUEST and RESPONSE

<table>
<thead>
<tr>
<th>CloudTransport Action(s)a,e</th>
<th>HTTP HEAD and DELETE</th>
<th>HTTP GET</th>
<th>HTTP POST and PUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPConnectMethod()</td>
<td>1b</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HTTPWriteHeaders()</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HTTPWriteBody()</td>
<td>0</td>
<td>0</td>
<td>[1 - N]</td>
</tr>
<tr>
<td>HTTPReadHeaders()</td>
<td>1c</td>
<td>1c</td>
<td>1</td>
</tr>
<tr>
<td>HTTPReadBody()</td>
<td>0</td>
<td>[0 to N]d</td>
<td>[0 to N]d</td>
</tr>
</tbody>
</table>

a The order of the actions, from top to bottom, indicates the order of execution to implement the HTTP REQUEST and RESPONSE indicated in the adjacent columns.

b This value indicates the number of invocations of the particular action typically needed to implement the HTTP REQUEST and RESPONSE indicated in the adjacent columns, where 0 indicates the action is not needed, 1 indicates it is needed once, N indicates more than 1, and bracketed values indicate low and high number of executions.

c If the HEADER RESPONSE is a "100 CONTINUE" then the next call(s) should be an HTTPReadHeaders() action until a non "100 CONTINUE" is received or the connection is closed.

d The control point should examine the RESPONSE HEADERS and determine if the transfer includes "TRANSFER-ENCODING: chunked" and/or "CONTENT-LENGTH" HEADERS and execute HTTPReadBody() actions as appropriate according to [HTTP1.1].

e Pipelines REQUESTs and RESPONSEs are allowed as long as the connection remains open. This is the default behaviour for HTTP 1.1 persistent connections.

6.6.2 HTTPConnectMethod()

This required action opens a connection between a local device, identified by the Host input argument and the CloudTransport service and sends an HTTP METHOD LINE described in the Method input argument to the UCCD for forming an HTTP REQUEST on the local network. Upon success it returns the Identifier output argument for the specific connection.
6.6.2.1 Arguments

Table 6-5 — Arguments for HTTPConnectMethod()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>IN</td>
<td>A_ARG_TYPE_Host</td>
</tr>
<tr>
<td>MethodLine</td>
<td>IN</td>
<td>A_ARG_TYPE_Headers</td>
</tr>
<tr>
<td>Identifier</td>
<td>OUT</td>
<td>A_ARG_TYPE_Identifier</td>
</tr>
<tr>
<td>ConnectWriteCount</td>
<td>OUT</td>
<td>A_ARG_TYPE_UI4</td>
</tr>
</tbody>
</table>

6.6.2.2 Argument Host

This input argument contains a string identifying a "host:port" value that the CloudTransport service should be able to connect to on its local (UDA) interface and exchange HTTP messages.

6.6.2.3 Argument MethodLine

This input argument contains an HTTP METHOD LINE according to [HTTP1.1]; For example:

"GET /MediaRenderer3.xml HTTP 1.1"

or

"HEAD /music/goodband/goodsong.mp3 HTTP 1.1".

6.6.2.4 Argument Identifier

This output argument provides a temporary, unique identifier that allows the CloudTransport service to distinguish all of its current connections to local devices back to the UCC-CPs using the CloudTransport service. It is recommended that it have a high degree of randomization to prevent guessing by other UCC-CPs.

6.6.2.5 Argument ConnectWriteCount

This output argument indicates the number of CloudTransport write related actions - HTTPConnectMethod(), HTTPWriterHeaders(), HTTPWriteBody() - transacted on the connection identified by Identifier since it was opened. It is a monotonically increasing value with the first transaction having a value of "1" and each subsequent transaction causing the value to increment by "1".

6.6.2.6 Service Requirements

This action is a Restrictable action as described in 4.2.

6.6.2.7 Control Point Requirements When Calling The Action

Successful invocation of the action on HTTPS connections is dependent on the control point Role or if the control point is a UCC-CP. See Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles for details.

6.6.2.8 Dependency on Device State

If the connection is already open between the UDA interface of the CloudTransport service and the local device for the same "host:port" combination then the Identifier value returned will be the same as previously returned for that connection.

6.6.2.9 Effect on Device State

None.
6.6.2.10 Errors

Table 6-6 — Error Codes for HTTPConnectMethod()

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>700</td>
<td>Not Connected</td>
<td>The connection failed.</td>
</tr>
<tr>
<td>701</td>
<td>Host not found</td>
<td>The indicated host could not be found</td>
</tr>
<tr>
<td>702</td>
<td>Invalid Connection</td>
<td>The connection indicated in Identifier is not recognized</td>
</tr>
<tr>
<td>703</td>
<td>Method in Queue</td>
<td>A METHOD LINE was already in the HTTP REQUEST buffer, flush or resend.</td>
</tr>
<tr>
<td>704</td>
<td>HTTPS not allowed</td>
<td>The control point is not allowed an HTTPS connection.</td>
</tr>
</tbody>
</table>

6.6.3 HTTPWriteHeaders()

This required action writes HTTP HEADER LINEs to an existing, open connection. It is highly recommended that it follow the successful invocation of a HTTPConnectMethod() action within 5 seconds.

6.6.3.1 Arguments

Table 6-7 — Arguments for HTTPWriteMethod()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>IN</td>
<td>A_ARG_TYPE_Identifier</td>
</tr>
<tr>
<td>Headers</td>
<td>IN</td>
<td>A_ARG_TYPE_Headers</td>
</tr>
<tr>
<td>ConnectWriteCount</td>
<td>OUT</td>
<td>A_ARG_TYPE_Ul4</td>
</tr>
</tbody>
</table>

6.6.3.2 Argument Identifier

This input argument identifies the HTTP connection that the HTTP REQUEST HEADER LINEs will be written to.

6.6.3.3 Argument Headers

This input argument contains the HTTP HEADER LINEs according to [HTTP1.1]; For example:

"HOST: 192.168.0.5:37222<CR>1<LF>2
CONNECTION: close<CR><LF>
USER-AGENT: 6.1.7601 2/Service Pack 1, UPnP/1.0, Portable SDK for UPnP devices/1.6.19<CR><LF><CR><LF>
"

It includes all <CR> and <LF> elements that would appear in the HEADER LINEs of a properly constructed HTTP REQUEST including the BLANK LINE. It shall not include an HTTP METHOD LINE. Its HOST HEADER LINE shall match the "host:port" value on the preceeding HTTPConnectMethod() action.

1 Carriage-Return will be HEX 0x0D and equals 1 byte of data on the wire.
2 Line-Feed will be HEX 0x0A and equals 1 byte of data on the wire.
6.6.3.4 Argument **ConnectWriteCount**

This output argument indicates the number of CloudTransport write related actions - `HTTPConnectMethod()`, `HTTPWriterHeaders()`, `HTTPWriteBody()` - transacted on the connection identified by `Identifier` since it was opened. It is a monotonically increasing value with the first transaction having a value of "1" and each subsequent transaction causing the value to increment by "1".

6.6.3.5 Service Requirements

This action is a **Restrictable** action as described in 4.2.

6.6.3.6 Control Point Requirements When Calling The Action

Successful invocation of the action on HTTPS connections is dependent on the control point `Role` or if the control point is a UCC-CP. See Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles for details.

6.6.3.7 Dependency on Device State

The connection indicated by `Identifier` shall be open to execute the action.

6.6.3.8 Effect on Device State

None.

6.6.3.9 Errors

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>700</td>
<td>Invalid Connection</td>
<td>The connection indicated in <code>Identifier</code> is not recognized</td>
</tr>
<tr>
<td>704</td>
<td>HTTPS not allowed</td>
<td>The control point is not allowed an HTTPS connection.</td>
</tr>
<tr>
<td>706</td>
<td>No METHOD LINE in Queue</td>
<td>A METHOD LINE is not in the HTTP REQUEST buffer, flush and send an HTTP METHOD LINE before sending HEADER LINES.</td>
</tr>
</tbody>
</table>

6.6.4 `HTTPReadHeaders()`

This required action reads the HTTP RESPONSE STATUS and HEADER LINES from an existing HTTP connection on the local network indicated by `Identifier`. It shall read to the first BLANK LINE encountered and include the BLANK LINE in the returned response.
6.6.4.1 Arguments

Table 6-9 — Arguments for HTTPReadHeaders()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>IN</td>
<td>A_ARG_TYPE_Identifier</td>
</tr>
<tr>
<td>CRLFFlag</td>
<td>IN</td>
<td>A_ARG_TYPE_Flag</td>
</tr>
<tr>
<td>Headers</td>
<td>OUT</td>
<td>A_ARG_TYPE_Headers</td>
</tr>
<tr>
<td>ReadLength</td>
<td>OUT</td>
<td>A_ARG_TYPE_Ui4</td>
</tr>
<tr>
<td>ConnectReadCount</td>
<td>OUT</td>
<td>A_ARG_TYPE_Ui4</td>
</tr>
</tbody>
</table>

6.6.4.2 Argument Identifier

This input argument identifies an existing HTTP connection that an HTTP STATUS LINE and RESPONSE HEADER LINEs will be read from.

6.6.4.3 Argument CRLFFlag

This input argument indicates if the action should return results after encountering a <CR><LF>. A value of "1" indicates to return the RESPONSE HEADER LINEs up to and including the first <CR><LF>. A value of "0" indicates to return the RESPONSE HEADER LINEs up to and including the first BLANK LINE.

6.6.4.4 Argument Headers

This output argument contains an HTTP RESPONSE string composed of the STATUS and HEADER LINEs including the <CR> and <LF> as a single string, for example, the HTTP RESPONSE to the HEAD METHOD

"HEAD /music/goodband/goodsong.mp3 HTTP 1.1",

would be

"HTTP/1.1 200 OK<CR><LF>
Date: Thu, 15 Oct 2015 21:37:54 GMT<CR><LF>
Content-Type: audio/mpeg<CR><LF>
Content-Length: 4001377<CR><LF>
contentFeatures: dlna.orgDLNA.ORG_PN=MP3;DLNA.ORG_OP=01;DLNA.ORG_FLAGS=01700000000000000000000000000000000000000000000000000<CR><LF>
Connection: close<CR><LF>
<CR><LF>"^1

It is the responsibility of the device supporting the CloudTransport service to parse and send only the HTTP RESPONSE STATUS and HEADER LINEs.

6.6.4.5 Argument ReadLength

This output argument shall indicate the actual length of HTTP RESPONSE HEADER LINEs data read and returned, in chars, including <CR><LF> and end of HEADERS "blank line". For example, in the example above the returned value of ReadLength would be 212.

^1 Note that some line formatting has been added to improve readability.
6.6.4.6 Argument ConnectReadCount

This output argument indicates the number of CloudTransport read related actions - HTTPReadHeaders() and HTTPReadBody() - transacted on the connection identified by Identifier since it was opened. It is a monotonically increasing value with the first transaction having a value of "1" and each subsequent transaction causing the value to increment by "1".

6.6.4.7 Service Requirements

This action is a Restrictable action as described in 4.2.

6.6.4.8 Control Point Requirements When Calling The Action

Successful invocation of the action on HTTPS connections is dependent on the control point Role or if the control point is a UCC-CP. See Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles for details.

6.6.4.9 Dependency on Device State

The connection indicated by Identifier shall be open to execute the action.

6.6.4.10 Effect on Device State

If the HTTP RESPONSE includes a "CONNECTION: Close" HEADER then the connection will be closed and the previously used Identifier deleted from the list of active connections at or before action invocation completion.

6.6.4.11 Errors

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>700</td>
<td>Invalid Connection</td>
<td>The connection indicated in Identifier is not recognized</td>
</tr>
<tr>
<td>704</td>
<td>HTTPS not allowed</td>
<td>The control point is not allowed an HTTPS connection.</td>
</tr>
<tr>
<td>707</td>
<td>No RESPONSE in Queue</td>
<td>Read RESPONSE does not appear to be RESPONSE STATUS and HEADER LINEs.</td>
</tr>
<tr>
<td>708</td>
<td>Connection closed prematurely</td>
<td>The connection was closed before the read could be completed.</td>
</tr>
</tbody>
</table>

6.6.5 HTTPReadBody()

This required action reads Size bytes of data from the HTTP RESPONSE BODY from the existing connection described by Identifier. The data shall be BASE64 encoded to make it suitable for inclusion in a UCA XMPP stanza. This means that the ratio of actual data to XMPP stanza payload will be 3 to 4 in most cases (BASE64 involves some padding when the data stream length is not divisible by 3).
6.6.5.1 Arguments

Table 6-11 — Arguments for HTTPReadBody()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>IN</td>
<td>A_ARG_TYPE_Identifier</td>
</tr>
<tr>
<td>Size</td>
<td>IN</td>
<td>A_ARG_TYPE_UI4</td>
</tr>
<tr>
<td>CRLFFlag</td>
<td>IN</td>
<td>A_ARG_TYPE_Flag</td>
</tr>
<tr>
<td>Body</td>
<td>OUT</td>
<td>A_ARG_TYPE_Body</td>
</tr>
<tr>
<td>ReadLength</td>
<td>OUT</td>
<td>A_ARG_TYPE_UI4</td>
</tr>
<tr>
<td>ConnectReadCount</td>
<td>OUT</td>
<td>A_ARG_TYPE_UI4</td>
</tr>
</tbody>
</table>

6.6.5.2 Argument Identifier

This input argument identifies the HTTP connection that a HTTP BODY data will be read from.

6.6.5.3 Argument Size

This input argument indicates the amount of BODY data to be read in bytes.

6.6.5.4 Argument CRLFFlag

This input argument indicates if the action should return results after encountering a \(<CR><LF>\). A value of "1" indicates to return the RESPONSE HEADER LINEs up to and including the first \(<CR><LF>\). A value of "0" indicates to return the RESPONSE HEADER LINEs up to nd including the first BLANK LINE.

6.6.5.5 Argument Body

This output argument contains BASE64 HTTP BODY data with length equivalent to Size when converted back from BASE64. The CloudTransport service does not try to interpret the data only convert it for UCA transmission. It is highly recommended to keep Size less than 6144 (6KB) to fit within typical XMPP stanza limitations. For example, if in the example in 6.6.4.4 instead of a HTTP HEAD METHOD REQUEST an HTTP GET METHOD REQUEST was made then after the HTTPReadHeaders() action was executed the HTTP BODY containing the media data would be obtained by issuing a series of HTTPReadBody() actions (roughly 652 invocations⁴) to retrieve the entire body.

For example a raw data stream (represented by the HEX equivalent below) would be converted

```
YTcwOWF1MzY5YjYxMTRhMzQ1ZTA5YzNlMDlkZDA0ZmZmYjkyNjBjYzgwMDU3MTVvmNTdkMzA2NzIzYTcwMGMfMyWUSMzE2NjZkMGQ3NTRiNWQwNzk4Y2RhMTI5YTI2Y2I0YjI4OTc0ZjliZGI4ODE4NDg2YTMyNzY3NDI0ZGE0YmY2c3cxMjIjg2MzVjNTViMjdiN2IyMTRmOWN2ZmZmZmNmF1OWE2ZGZkYjRhYWZlZGJjYzY0T1hNmJiYWE4MzY4YjgwODYmYjJhNhNzkyM2IzM2MmMTKzZWJhnMThiMjVjYjAwMjRjYjNjkyNmUzZmZkZGEwMDJ1NDVkNGJjOTA5YmIyYTYyMzI5Y2U1Ny
```

to its BASE64 equivalent:

```
"a709ae369b6114a345e09c3e09d004fffb9260cc8005715f57d3067b6a700a9e931666d0d754b5d0798cda129a26cb4b28974f9b9db8818486a310f7424da4a71fbb2863ec55b27e7b21345ffff6ae9a6dfdb4aab6dbcc64ae9a6bbab8368b8090fb2a7923b33cf193eba18b25cb0024c6926e5ffdda102e45d4bc09bb2a6323a565f1325ced1d1279b1193c5901d54f289213d893a471a95174d0a867fa534abe3b259890a4af37ef9d"
```

⁴ 4001377 ÷ 6144 = 651.2677
to generate the **Body** output argument.

### 6.6.5.6 Argument **ReadLength**

This output argument shall indicate the actual length of HTTP BODY data read and returned, in bytes, prior to conversion to BASE64 encoding. For example, the returned value of **ReadLength** would be 6144 if the local HTTP client's buffer is full.

### 6.6.5.7 Argument **ConnectReadCount**

This output argument indicates the number of CloudTransport read related actions - `HTTPReadHeaders()` and `HTTPReadBody()` - transacted on this connection since it was opened. It is a monotonically increasing value with the first transaction having a value of "1" and each subsequent transaction causing the value to increment by "1".

### 6.6.5.8 Service Requirements

This action is a **Restrictable** action as described in 4.2.

### 6.6.5.9 Control Point Requirements When Calling The Action

Successful invocation of the action on HTTPS connections is dependent on the control point **Role** or if the control point is a UCC-CP. See Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles for details.

The control point issuing the `HTTPReadBody()` actions shall calculate the number of bytes needed to be read by using either the HEADER LINE "Content-Length" value or in-stream "Chunked" length values.

### 6.6.5.10 Dependency on Device State

The connection indicated by **Identifier** shall be open to execute the action.

### 6.6.5.11 Effect on Device State

None unless the connection is closed.

### 6.6.5.12 Errors

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>700</td>
<td>Invalid Connection</td>
<td>The connection indicated in <strong>Identifier</strong> is not recognized</td>
</tr>
<tr>
<td>704</td>
<td>HTTPS not allowed</td>
<td>The control point is not allowed an HTTPS connection.</td>
</tr>
<tr>
<td>708</td>
<td>Connection closed prematurely</td>
<td>The connection was closed before the read was completed.</td>
</tr>
</tbody>
</table>

### 6.6.6 **HTTPWriteBody()**

This required action writes HTTP BODY **Size** bytes of data to an existing connection described by **Identifier**. The data shall be BASE64 encoded to make it suitable for inclusion in a UCA XMPP stanza. This means that the ratio of actual data to XMPP stanza payload will be 3 to 4.
6.6.6.1 Arguments

Table 6-13 — Arguments for HTTPWriteBody()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>IN</td>
<td>A_ARG_TYPE_Identifier</td>
</tr>
<tr>
<td>Body</td>
<td>IN</td>
<td>A_ARG_TYPE_Body</td>
</tr>
<tr>
<td>Size</td>
<td>IN</td>
<td>A_ARG_TYPE_UI4</td>
</tr>
<tr>
<td>ConnectWriteCount</td>
<td>OUT</td>
<td>A_ARG_TYPE_UI4</td>
</tr>
</tbody>
</table>

6.6.6.2 Argument **Identifier**

This input argument provides a temporary, unique identifier that allows the CloudTransport service to distinguish all of its current connections to a local device to UCC-CPs using the CloudTransport service.

6.6.6.3 Argument **Body**

This input argument contains BASE64 HTTP BODY data with length equivalent to **Size** when converted back from BASE64.

6.6.6.4 Argument **Size**

This input argument indicates the amount of BODY data to be written in bytes when converted back from BASE64.

6.6.6.5 Argument **ConnectWriteCount**

This output argument indicates the number of CloudTransport write related actions - HTTPConnectMethod(), HTTPWriterHeaders(), HTTPWriteBody() - transacted on the connection identified by **Identifier** since it was opened. It is a monotonically increasing value with the first transaction having a value of "1" and each subsequent transaction causing the value to increment by "1".

6.6.6.6 Service Requirements

This action is a **Restrictable** action as described in 4.2.

6.6.6.7 Control Point Requirements When Calling The Action

Successful invocation of the action on HTTPS connections is dependent on the control point **Role** or if the control point is a UCC-CP. See Table 4-1 — Assignment of Restrictable/Non-Restrictable Roles for details.

The control point issuing the HTTPWriteBody() actions shall keep track of the number of bytes sent and make sure it corresponds to the HEADER LINE "Content-Length" value or in-stream "Chunked" length values sent in respective HTTPWriteHeaders() actions.

6.6.6.8 Dependency on Device State

The connection indicated by **Identifier** shall be open to execute the action.

6.6.6.9 Effect on Device State

None unless the connection is closed.
6.6.6.10 Errors

Table 6-14 — Error Codes for HTTPWriteBody()

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture clause on Control.</td>
</tr>
<tr>
<td>700</td>
<td>Invalid Connection</td>
<td>The connection indicated in Identifier is not recognized</td>
</tr>
<tr>
<td>704</td>
<td>HTTPS not allowed</td>
<td>The control point is not allowed an HTTPS connection.</td>
</tr>
<tr>
<td>708</td>
<td>Connection closed prematurely</td>
<td>The connection was closed before the write could be completed.</td>
</tr>
</tbody>
</table>

7 THEORY OF OPERATIONS (INFORMATIVE)

This section describes several typical scenarios encountered where the CloudTransport service is utilized. The device supporting the CloudTransport service will provide socket and TCP support, including TCP keep-alives on its local interface where the HTTP Server and Client will run. It will also need to include buffering to manage HTTP traffic over the local connection for fragmentation and reassemble to the CloudTransport actions.

7.1 GENERAL USAGE SCENARIOS

7.1.1 HTTP HEAD REQUEST and RESPONSE

In the first scenario the UCC-CP wants to execute a HTTP HEAD REQUEST to get the size of a media item used in a previous example. This will require the sequence of action invocations of HTTPConnectMethod() followed by HTTPWriteHeaders() and HTTPReadHeaders(). The transaction flow is illustrated in Figure 7-1 and described afterwards.
Figure 7-1 HTTP HEAD REQUEST and RESPONSE

- HTTP File Server with <res> on UDA Inf
- MediaServer with Cloud Transport Service
  - UDA Inf
- MediaServer with Cloud Transport Service
  - UCCD
- UMC
- UCC-GP
- UCC-GP

Start of HTTP Session

- The CloudTransport service opens a connection to the HTTP server
- Connection keep-alive(s) as needed
- The CloudTransport service sends the HTTP HEAD REQUEST to the HTTP server
- The HTTP Server sends the HTTP HEAD RESPONSE back to the CloudTransport service and closes the connection

End of HTTP Session

- HTTP File Server with <res> on UDA Inf
- MediaServer with Cloud Transport Service
  - UDA Inf
- MediaServer with Cloud Transport Service
  - UCCD
- UMC
- UCC-GP
- UCC-GP

HTTPConnectMethod()}

The UCC-GP invokes an HTTPConnectMethod() action with an HTTP HEAD REQUEST

HTTPWriteHeader()}

The UCC-GP invokes an HTTPWriteHeader() action

HTTPReadHeader()}

The CloudTransport service sends the HTTP HEAD RESPONSE to the UCC-GP from its internal buffer

HTTPReadHeader()}

The UCC-GP invokes an HTTPReadHeader() action
7.1.1.1 Opening a Connection with **HTTPConnectMethod()**

A UCC-CP knows a `<res>` is available at "http://192.168.0.5/music/goodband/goodsong.mp3" and wants to use an HTTP HEAD REQUEST to get any additional information, such as, the resources size. It invokes the **HTTPConnectMethod()** action as follows:

**Request (SOAP/XMPP):**

```
HTTPConnectMethod(  
  192.168.0.5:37222,  
  HEAD /music/goodband/goodsong.mp3 HTTP 1.1  
)
```

The UCCD first tries to open a TCP connection at "192.168.0.5:37222". Upon success, it creates a unique identifier of the specific connection for the **Identifier** output argument and since this port was just opened assigns a **ConnectWriteCount** value of "1" to this connection. The CloudTransport service will most likely buffer the HEAD METHOD LINE as it is expecting the HTTP HEADER LINEs in the subsequent **HTTPWriteHeaders()** action. The device supporting the CloudTransport service is responsible for keeping the connection alive while the UCC-CP builds the complete HTTP REQUEST stream.

**Response (SOAP/XMPP):**

```
HTTPConnectMethod(  
  "MxMGY3NDI0ZGE0YTcxZmJiMjg2M2VjNTViMjd1N2IyMTM",  
  "1"  
)
```

7.1.1.2 Completing the HTTP REQUEST with **HTTPWriteHeaders()**

The UCC-CP then completes the HTTP HEAD REQUEST by supplying the HTTP HEADER LINEs to the CloudTransport service for the previously opened connection as follows.

**Request (SOAP/XMPP):**

```
HTTPWriteHeaders(  
  "MxMGY3NDI0ZGE0YTcxZmJiMjg2M2VjNTViMjd1N2IyMTM",  
  HOST: 192.168.0.5:37222<CR><LF>  
  CONNECTION: close<CR><LF>  
  USER-AGENT: 6.1.7601 2/Service Pack 1, UPnP/1.0, Portable SDK for UPnP devices/1.6.19<CR><LF>  
  <CR><LF>  
)
```

The CloudTransport service knowing that the HTTP HEAD REQUEST is complete then sends the HTTP REQUEST on its local connection to the HTTP Server and more than likely receive the HTTP RESPONSE in its local connection before the UCC-CP issues the expected **HTTPReadHeaders()** action. Since this is the second write to this connection it will also increment the **ConnectWriteCount** output argument for this connection to "2". Below is the response from the action.

**Response (SOAP/XMPP):**

```
HTTPWriteHeaders(2)  
```

Note that even though the connection may have been closed after the HTTP RESPONSE was sent on the local network this is not indicated until after the buffer on the CloudTransport service side has been read by an **HTTPReadHeaders()** action or an implementation defined timeout has occurred.
7.1.1.3 Completing the HTTP RESPONSE with \textit{HTTPReadHeaders()}

The UCC-CP is now expecting a response from the HTTP Server on the local network so it issues an \textit{HTTPReadHeaders()} action on the connection that it has written the HTTP RESPONSE on as follows:

\textbf{Request (SOAP/XMPP):}
\begin{verbatim}
HTTPReadHeaders(
    MxMGY3NDI0ZGE0YTcxZmJiMjg2M2VjNTViMjd1N2IyMTM,
    0)
\end{verbatim}

The CloudTransport service reads the HTTP RESPONSE LINES from its connection buffer increments the ConnectReadCount to "1" since this is the first read on the connection, and returns the HTTP RESPONSE as follows:

\textbf{Response (SOAP/XMPP)}
\begin{verbatim}
HTTPReadHeaders(
    HTTP/1.1 200 OK<CR><LF>
    Date: Thu, 15 Oct 2015 21:37:54 GMT<CR><LF>
    Content-Type: audio/mpeg<CR><LF>
    Content-Length: 4001377<CR><LF>
    contentFeatures:
    dlna.orgDLNA.ORG_PN=MP3;DLNA.ORG_OP=01;
    DLNA.ORG_FLAGS=01700000000000000000000000000000<CR><LF>
    Connection: close<CR><LF>
    <CR><LF>,
231,
1)
\end{verbatim}

The UCC-CP should recognize that the connection has been closed by examining the HTTP RESPONSE message and not try to re-use the previous \textit{Identifier}.

7.1.2 HTTP GET REQUEST and RESPONSE

In the second scenario the UCC-CP wants to execute a HTTP GET REQUEST to retrieve the media item from the previous example. This will require the sequence of action invocations of \textit{HTTPConnectMethod()} followed by \textit{HTTPWriteHeaders()}, \textit{HTTPReadHeaders()} and then a series of \textit{HTTPReadBody()} actions. The transaction flow is illustrated in Figure 7-2 and Figure 7-3 and described afterwards.
Figure 7-2 HTTP GET REQUEST and RESPONSE Part 1

The CloudTransport service opens a connection to the HTTP server.

Connection keep-alive(s) as needed.

The CloudTransport service sends the HTTP GET REQUEST to the HTTP server.

The HTTP Server sends the HTTP GET RESPONSE back to the CloudTransport service including HEADERS and BODY.

The HTTP Server opens a connection to the HTTP Request.

The UCC-CP involves an HTTPConnectMethod() action with a HTTP GET REQUEST.

HTTPConnectMethod()}

HTTPConnectMethod()}

HTTPConnectMethod()}

The UCC-CP involves an HTTPWriteHeader() action.

HTTPWriteHeaders()}

HTTPWriteHeaders()}

HTTPWriteHeaders()}

The UCC-CP involves an HTTPReadHeader0 action.

HTTPReadHeaders()}

HTTPReadHeaders()}

HTTPReadHeaders()}

The CloudTransport service pulls the HTTP HEADERS from its internal buffer and sends to the UCC-CP.

HTTPReadHeaders()}

HTTPReadHeaders()}

HTTPReadHeaders()}

BODY Transfer Begins.

Copyright © 2015 Open Connectivity Foundation, Inc. All rights reserved.
Figure 7-3 HTTP GET REQUEST and RESPONSE Part 2

The HTTP RESPONSE will occur over a continued period. The CloudTransport service may need to use techniques to avoid buffer overflow.

The BODY transfer is likely to close before all the BODY is read and the connection close

The UCC-CP invokes an HTTPReadBody() action of fixed length

HTTP Read Body (N Bytes)

The CloudTransport service pulls N Bytes of BODY data from its internal buffer

HTTP Read Body (N Bytes)

The CloudTransport service converts the raw data to BASE64 before sending to UCC-CP

HTTP Read Body (N Bytes)

1 to N invocations of HTTPReadBody() occurs

HTTP Read Body (Last Bytes)

HTTP Read Body (Last Bytes)
7.1.2.1 The HTTP GET REQUEST

The HTTP GET REQUEST is formed in almost the identical fashion as the HTTP HEAD request described in sections 0 and 7.1.1.2 and the HTTP RESPONSE HEADER line retrieval executed as described in section 07.1.1.3. The main difference is in the actual METHOD LINE. For this example, the HTTPConnectMethod(), HTTPWriteHeaders(), and HTTPReadHeaders() actions have already been executed.

7.1.2.2 The HTTP BODY RESPONSE

In this example the HTTP RESPONSE is not Chunked and the HTTP Server will start sending the BODY to the CloudTransport service over its local connection. This data will need to be buffered in the CloudTransport service since the local interface will usually be much faster the UCA interface. The local connection will have to be managed to keep the buffer from overflowing using lower layer techniques such as TCP throttling. If the buffer of the CloudTransport service is large enough and the link fast enough, the complete BODY of the requested resource could be buffered in the CloudTransport service host device before any significant part of it has been sent to the UCC-CP.

The UCC-CP will know that the next data in the connection will be the BODY data since it successfully invoked the HTTPReadHeaders() action just previous to the invocation of the first HTTPReadBody() action as shown next. The UCC-CP requests that CloudTransport service retrieve 6000 bytes of data from the HTTP stream (this will actually be 8000 bytes of BASE64 encoded stanza data).

**Request (SOAP/XMPP):**

```
HTTPReadBody(
    2VjNTViMjdlN2MxMGY3NDI0ZGIyMTMe0YTcxZmJiMjg2M,
    6000,
    0
)
```

The CloudTransport service pulls 6000 bytes from its buffer (or to EOF),

```
0023565c61690023542e7ca9080450005dced45000406e66ac0a80005c0a801684
89f74f37af2ceb09c4ac501000ed3c60004944330300000020f76415044930000
1f4f800000696d61676526fa7067000000f8d8ffe000104a464946000101000010
... 
```

converts the data to BASE64 and sends the response as follows:

**Response (SOAP/XMPP):**

```
HTTPReadBody(
    MDAyMzU2NWM2MTY5MDAyMzU0MmU3Y2E5MDgwMDQ1MDAwNWRjY2Q0NTQwMDA0MDA2ZTY2YWml
    YTwgMDAyMzBhODAwMTY4NGFhZmMwZmMzM2VmMmMnY1YjA5Y2NhMQ1MDAwMDAwMDB1ZGQxYW
    MlM0NDMzMDMwMDAwMDAwMjBmNzYMTUwNDk0MzAwMDAxZjQ4MDAwMDAwNjk2ZDYxNjc2NTJm
    N...
    GM3MzkyNjkwJzVlZTU0ODdlMTFkZGMxYTItZTU5YWJhYmRiMTE3NzZmNmIxMzQ4NjA2MmNhZ
    DB1NjcyMzliODdkM2IzYWRlZDriYzBiNzVhOTdmNjY0Yzc1MDgyM2I5ZDM2ZTk2NmI3NmYyY
    2I4Mljl2YW1N2NmMmM1Yg==,
    6000,
    2
)
```

The UCC-CP continues to issue the HTTPReadBody() action until it has transferred the complete 4001377 bytes indicated in the "Content-Length" HEADER.

The last HTTPReadBody() action would look like this:
Request (SOAP/XMPP):

HTTPReadBody(
    2VjNTViMjd1N2MxMGY3NDI0ZGIyMTME0YTcxZmJiMjg2M,
    5377,
    0
)

with response

Response (SOAP/XMPP):

HTTPReadBody(
    . . .
    AwMDAwNDg2OTczNzQ2ZjcyNzkzYTiwNDE2ZDY1NzI2OTYzNjEyNzczMjA0NzcyNjU2MTc0NjU3Mzc0MjA0ODY5MzEzOTM3MzUyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDlN2MxMGY3NDI0ZGIyMTME0YTcxZmJiMjg2M,
    5377,
    668
)

7.1.3 HTTP GET REQUEST with "Chunked" RESPONSE

If the HTTP RESPONSE includes the "Transfer-Encoding: chunked" HEADER and there is an absence of a "Content-Length" HEADER then the UCC-CP should expect the BODY to be "chunked" and will need to use the HTTPReadBody() action(s) appropriately.

In this "chunked" example, the BODY data is sent in 3 "chunks" of length 60 bytes, 60 bytes, and 30 bytes (plus the zero bytes line to end the "chunking"). On the wire this would look like:

- 0x32<CR><LF> or the 6 octets "320D0A" converted to BASE64 (or "MzIwRDBB")
- [60 Bytes of Data] converted to BASE64 (80 Bytes in the Body payload)
- 0x32<CR><LF> or the 6 octets "320D0A" converted to BASE64 (or "MzIwRDBB")
- [60 Bytes of Data] converted to BASE64 (80 Bytes in the Body payload)
- 0x1A<CR><LF> or "160D0A" converted to BASE64 (or "MzIwRDBB")
- [30 Bytes of Data] converted to BASE64 (40 Bytes in the Body payload)
- 0x0<CR><LF> "000D0A" or converted to BASE64 (or "MDBEMEE=")

A conservative strategy is used to get the BODY data using 7 HTTPReadBody() actions. The first, third, fifth, and seventh are used to nibble the chunk sizes, the, second, fourth, and sixth are used to return the BODY data. The invocations for the first two actions are shown as follows:

First HTTPBodyRead() to get length with CRLFFlag set to "1"

Request (SOAP/XMPP):

HTTPReadBody(
    Mjd1N2MxMGY3NDI0ZGIyMT2VjNTVi,
    500,
    1
)

with response

Response (SOAP/XMPP):
HTTPReadBody(
    MzIwRDBB,
    6,
    725
)

and second **HTTPReadBody()** to get first BODY data chunk with **CRLFFlag** set to "0"

**Request (SOAP/XMPP):**

HTTPReadBody(
    MjdIN2MxMGY3NDI0ZGIyMT2VjNTV1,
    60,
    0
)

with response

**Response (SOAP/XMPP):**

HTTPReadBody(
    MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIz
    NDU2Nzg5,
    60,
    726
)

MzIwRDBB

### 7.1.4 HTTP POST REQUEST with "100-Continue"

When sending an HTTP POST REQUEST to the HTTP Server with an "EXPECT: 100-Continue" HEADER LINE included, the Client (UCC-CP) should issue a **HTTPReadHeaders()** action to get the "HTTP/1.1 100 CONTINUE" HTTP STATUS or other HTTP status message before sending the BODY data for the POST REQUEST using the **HTTPWriteBody()** action.

#### 7.1.5 Closing an HTTP Connection

There is no explicit action for closing an HTTP connection instead a normal HTTP "Connection: close" HEADER is used with the HTTP Server closing the connection after sending the HTTP RESPONSE and the HTTP Client, in the device, closing after receiving the last RESPONSE.

### 8 XML SERVICE DESCRIPTION

```xml
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <actionList>
        <action>
            <name>HTTPConnectMethod</name>
            <argumentList>
                <argument>
                    <name>Host</name>
                    <direction>in</direction>
                    <relatedStateVariable>
                        A_ARG_TYPE_Host
                    </relatedStateVariable>
                </argument>
                <argument>
                    <name>MethodLine</name>
                </argument>
            </argumentList>
        </action>
    </actionList>
</scpd>
```
<action>
  <name>HTTPWriteHeaders</name>
  <argumentList>
    <argument>
      <name>Identifier</name>
      <direction>in</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Identifier
      </relatedStateVariable>
    </argument>
    <argument>
      <name>Headers</name>
      <direction>in</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Headers
      </relatedStateVariable>
    </argument>
    <argument>
      <name>ConnectWriteCount</name>
      <direction>out</direction>
      <relatedStateVariable>
        A_ARG_TYPE_UI4
      </relatedStateVariable>
    </argument>
  </argumentList>
</action>

<action>
  <name>HTTPReadHeaders</name>
  <argumentList>
    <argument>
      <name>Identifier</name>
      <direction>in</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Identifier
      </relatedStateVariable>
    </argument>
    <argument>
      <name>CRLFFlag</name>
      <direction>in</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Flag
      </relatedStateVariable>
    </argument>
    <argument>
      <name>Headers</name>
      <direction>out</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Headers
      </relatedStateVariable>
    </argument>
  </argumentList>
</action>
<argument>
  <name>ReadLength</name>
  <direction>out</direction>
  <relatedStateVariable>
    A_ARG_TYPE_UI4
  </relatedStateVariable>
</argument>
<argument>
  <name>ConnectReadCount</name>
  <direction>out</direction>
  <relatedStateVariable>
    A_ARG_TYPE_UI4
  </relatedStateVariable>
</argument>
</argumentList>
</action>
</action>
<action>
  <name>HTTPReadBody</name>
  <argumentList>
    <argument>
      <name>Identifier</name>
      <direction>in</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Identifier
      </relatedStateVariable>
    </argument>
    <argument>
      <name>Size</name>
      <direction>in</direction>
      <relatedStateVariable>
        A_ARG_TYPE_UI4
      </relatedStateVariable>
    </argument>
    <argument>
      <name>CRLFFlag</name>
      <direction>in</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Flag
      </relatedStateVariable>
    </argument>
    <argument>
      <name>Body</name>
      <direction>out</direction>
      <relatedStateVariable>
        A_ARG_TYPE_Body
      </relatedStateVariable>
    </argument>
    <argument>
      <name>ReadLength</name>
      <direction>out</direction>
      <relatedStateVariable>
        A_ARG_TYPE_UI4
      </relatedStateVariable>
    </argument>
    <argument>
      <name>ConnectReadCount</name>
      <direction>out</direction>
      <relatedStateVariable>
        A_ARG_TYPE_UI4
      </relatedStateVariable>
    </argument>
  </argumentList>
</action>
<action>
  <name>HTTPWriteBody</name>
<argumentList>
  <argument>
    <name>Identifier</name>
    <direction>in</direction>
    <relatedStateVariable>
      A_ARG_TYPE_Identifier
    </relatedStateVariable>
  </argument>
  <argument>
    <name>Body</name>
    <direction>in</direction>
    <relatedStateVariable>
      A_ARG_TYPE_Body
    </relatedStateVariable>
  </argument>
  <argument>
    <name>Size</name>
    <direction>in</direction>
    <relatedStateVariable>
      A_ARG_TYPE_UI4
    </relatedStateVariable>
  </argument>
  <argument>
    <name>ConnectWriteCount</name>
    <direction>out</direction>
    <relatedStateVariable>
      A_ARG_TYPE_UI4
    </relatedStateVariable>
  </argument>
</argumentList>
</actionList>
</serviceStateTable>