CloudTransport:1 Service

For UPnP Version 2.0

Status: Standardized DCP (SDCP)

Date: December 31, 2015

Document Version: 1.0

Service Template Version: 2.00

This Standardized DCP has been adopted as a Standardized DCP by the Steering Committee of the UPnP Forum, pursuant to Section 2.1(c)(ii) of the UPnP Forum Membership Agreement. UPnP Forum Members have rights and licenses defined by Section 3 of the UPnP Forum Membership Agreement to use and reproduce the Standardized DCP in UPnP Compliant Devices. All such use is subject to all of the provisions of the UPnP Forum Membership Agreement.

THE UPNP FORUM TAKES NO POSITION AS TO WHETHER ANY INTELLECTUAL PROPERTY RIGHTS EXIST IN THE STANDARDIZED DCPS. THE STANDARDIZED DCPS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". THE UPNP FORUM MAKES NO WARRANTIES, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE STANDARDIZED DCPS, INCLUDING BUT NOT LIMITED TO ALL IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, OF REASONABLE CARE OR WORKMANLIKE EFFORT, OR RESULTS OR OF LACK OF NEGLIGENCE.

Authors [a]	**Company**
Clarke Stevens	Cablelabs
Wouter van der Beek (Chair)	Cisco
Bich Nguyen	GoPro
Keith Miller	BKM Systems Group, LLC

[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

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.

[UDA] UPnP Device Architecture, version 2.0, UPnP Forum, February 20, 2015. Available at: http://upnp.org/specs/arch/UPnPDA10_20000613.pdf. Latest version available at: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf.

[ISO_8601] ISO 8601 Data elements and interchange formats – Information interchange -- Representation of dates and times, International Standards Organization, December 21, 2000. Available at: http://www.iso.org (ISO 8601:2004).

[CDS4] UPnP Content Directory Service, version 4.0, UPnP Forum, June 30, 2015. Available at http://www.upnp.org/specs/av/ContentDirectory-av-v4-Service-201150630.pdf. Latest version available at: http://www.upnp.org/specs/av/ContentDirectory-av-v4-Service.pdf.

[RFC_2119] IETF RFC 2119, Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, 1997. Available at: http://www.faqs.org/rfcs/rfc2119.html.

[HTTP1.1] HyperText Transport Protocol – HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999. Available at: http://www.ietf.org/rfc/rfc2616.txt.

[RFC_3339] IETF RFC 3339, Date and Time on the Internet: Timestamps, G. Klyne, Clearswift Corporation, C. Newman, Sun Microsystems, July 2002. Available at: http://www.ietf.org/rfc/rfc3339.txt.

[RFC_6122] IETF RFC 6122, Extensible Messeging and Presence Protocol, P. Saint-Andre, Cisco, March, 2011. Available at: http://www.ietf.org/rfc/rfc6122.txt.

[XML 1.0] Extensible Markup Language (XML) 1.0 (Third Edition), François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, eds., W3C Recommendation, February 4, 2004. Available at: http://www.w3.org/TR/2004/REC-xml-20040204.

[XSD 2.0] XML Schema Part 2: Data Types, Second Edition, Paul V. Biron, Ashok Malhotra, W3C Recommendation, 28 October 2004. Available at: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028.

[PROXY] UPnP CloudProxy:1 Device, UPnP Forum July 1, 2013. Available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudProxy-v1-Device-20130701.pdf. Latest version available at: http://www.upnp.org/specs/smgt/UPnP-smgt-SensorManagement-v1-Device.pdf.

[DP] UPnP DeviceProtection:1 Service, UPnP Forum, February 24, 2011. Available at: http://www.upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service-

20110224.pdf. Latest version available at: http://www.upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service.pdf.

[CPROXY] UPnP CloudProxy:1 Service, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudProxy-v1-Service-20151231.pdf. Latest version available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudProxy-v1-Service.pdf.

[CTS] UPnP CloudTransport:1 Service, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudTransport-v1-Service-20151231.pdf. Latest version available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudTransport-v1-Service.pdf.

[XSD_CPU] XML Schema UPnP CloudProxy Update IDs, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/schemas/cloud/cloudproxyupdate-v1-20151231.xsd. Latest version available at: http://www.upnp.org/schemas/cloud/cloudproxyupdate.xsd.

[XSD_CPDL] XML Schema UPnP CloudProxy Device List, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/schemas/cloud/devicelist-v1-20151231.xsd. Latest version available at: http://www.upnp.org/schemas/cloud/devicelist.xsd.

[XSD_CPPL] XML Schema UPnP CloudProxy Proxy List, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/schemas/cloud/proxylist-v1-20151231.xsd. Latest version available at: http://www.upnp.org/schemas/cloud/proxylist.xsd.

[XSD_CPUL] XML Schema UPnP CloudProxy UCS List, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/schemas/cloud/ucslist-v1-20151231.xsd. Latest version available at: http://www.upnp.org/schemas/cloud/ucslist.xsd.

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 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

Action Name	**Restrictable/Non-Restrictable to Indicated Role**[1]				
	No DeviceProtection	Public	Basic	Admin	UCC-CP
HTTPConnectMethod()	HTTP	HTTP	HTTP, HTTPS	HTTP, HTTPS	HTTP, HTTPS
HTTPReadHeaders()	HTTP	HTTP	HTTP, HTTPS	HTTP, HTTPS	HTTP, HTTPS
HTTPWriteHeaders()	HTTP	HTTP	HTTP, HTTPS	HTTP, HTTPS	HTTP, HTTPS
HTTPReadBody()	HTTP	HTTP	HTTP, HTTPS	HTTP, HTTPS	HTTP, HTTPS
HTTPWriteBody()	HTTP	HTTP	HTTP, HTTPS	HTTP, HTTPS	HTTP, HTTPS

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.

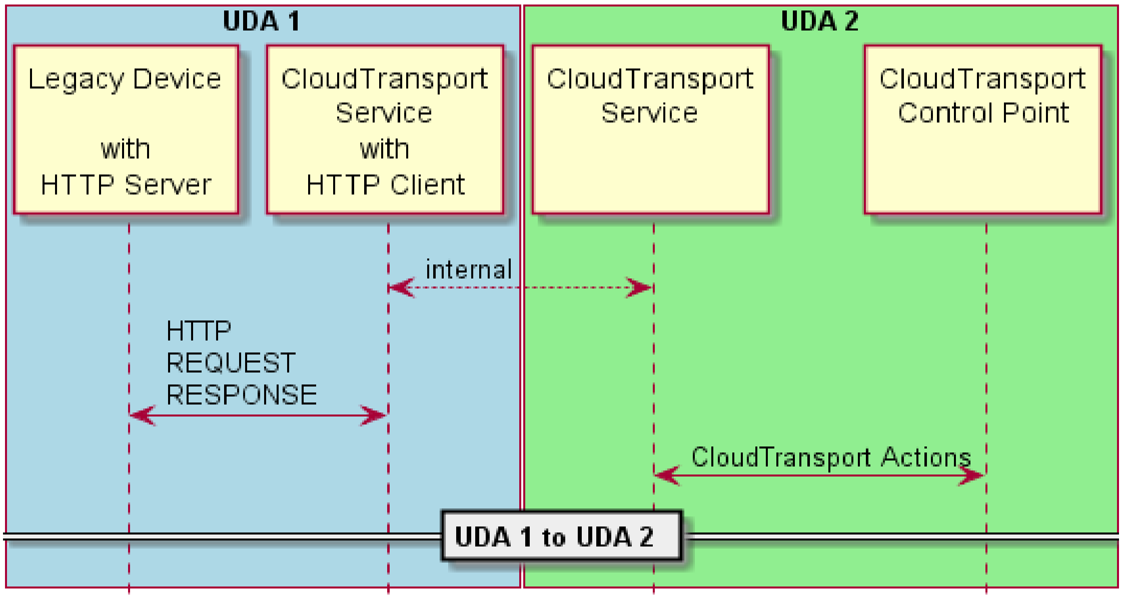
6.2 CLOUDTRANSPORT:1 SERVICE ARCHITECTURE

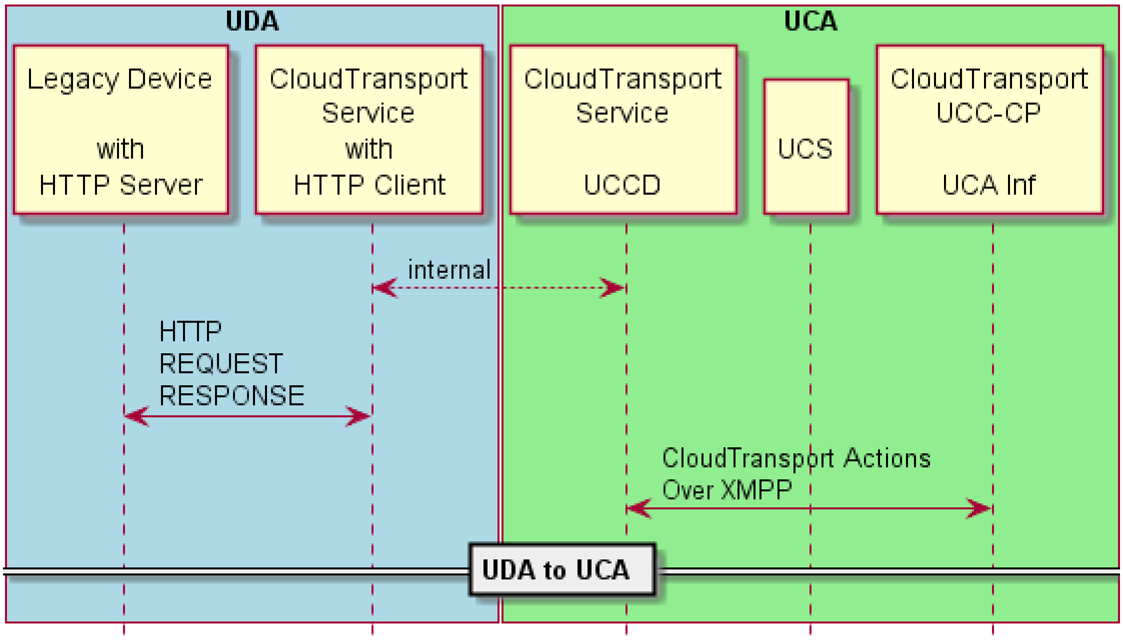
The CloudTransport service consists of these architectural elements:

- 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.

Figure 6-1 CloudTransport Architecture UDA-to-UDA and UDA-to-UCA


UDA 1
UDA 2
Legacy Device with HTTP Server
CloudTransport Service with HTTP Client
CloudTransport Service
CloudTransport Control Point
internal
HTTP REQUEST RESPONSE
CloudTransport Actions
UDA 1 to UDA 2


UDA
UCA
Legacy Device with HTTP Server
CloudTransport Service with HTTP Client
CloudTransport Service UCCD
UCS
CloudTransport UCC-CP UCA Inf
internal
HTTP REQUEST RESPONSE
CloudTransport Actions Over XMPP
UDA to UCA

6.3 KEY CONCEPTS

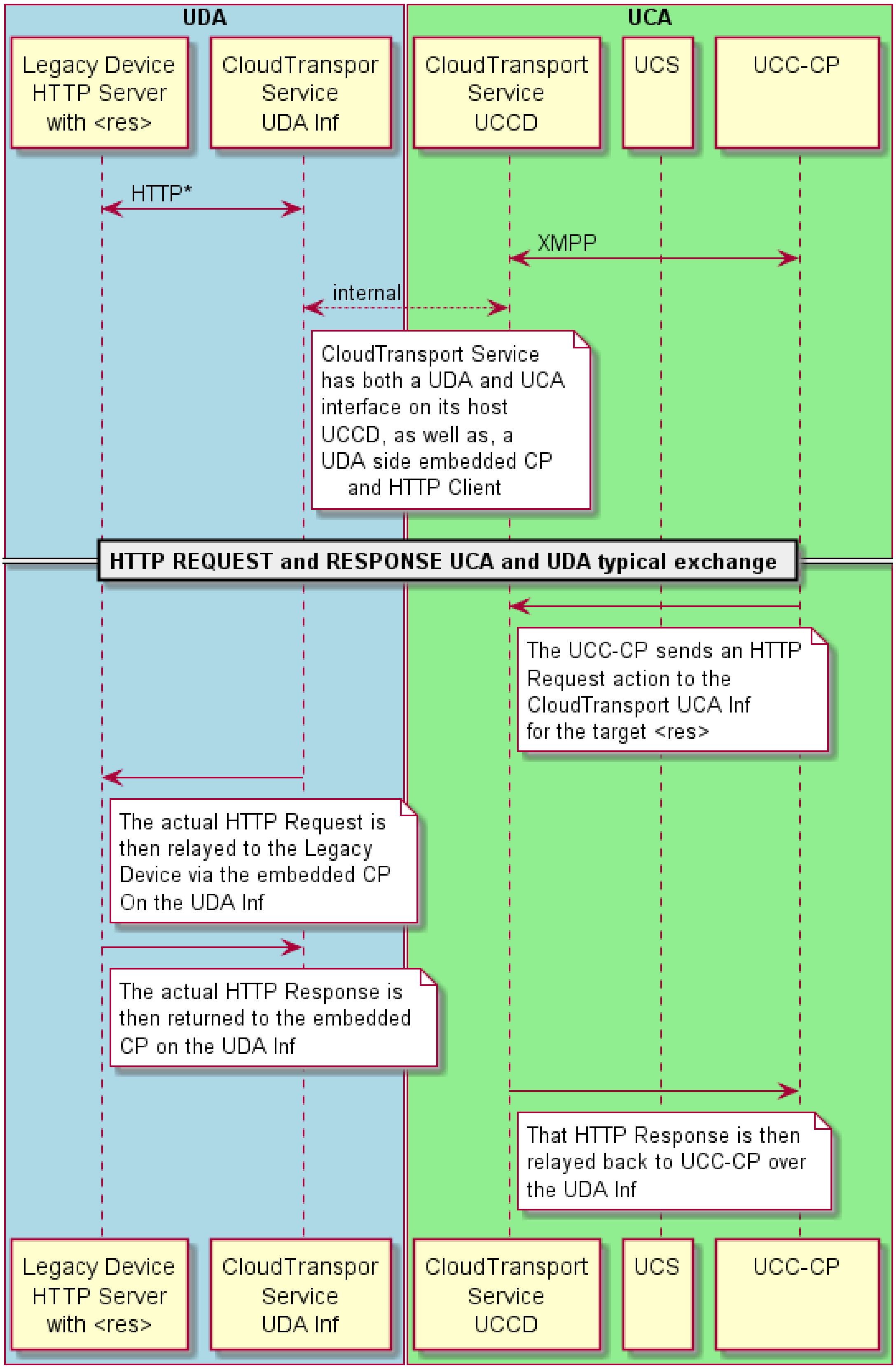
The CloudTransport service provides the functionality to relay HTTP REQUESTs and RESPONSEs over a remote endpoint. It is suitable for usage over a UCA interface, as well as, a UDA (local) interface; although it could be used to connect UDA devices on the same local network, or different local networks, 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


UDA
UCA
Legacy Device HTTP Server with <res>
CloudTranspor Service UDA Inf
CloudTransport Service UCCD
UCS
UCC-CP
HTTP*
XMPP
internal
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 UCC-CP sends an HTTP Request action to the CloudTransport UCA Inf for the target <res>
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
That HTTP Response is then relayed back to UCC-CP over the UDA Inf
Legacy Device HTTP Server with <res>
CloudTranspor Service UDA Inf
CloudTransport Service UCCD
UCS
UCC-CP

* This connection may be of type linklocal, that is, the <res> item is on the same device as the CloudTransport service or between two devices in the local network, for example, one device is hosting an HTTP file server and the other is a UCCD MediaServer or a CloudProxy device [PROXY].

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 6-1 — State Variables

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

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_Headers*

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 6-2 — Eventing and Moderation

Variable Name	Evented	Moderation	
		Moder-ated [a]	Criteria

[a] *YES* = The state variable shall be moderated with the criteria

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

Name	Device R/A [a]	Control Point R/A [b]
HTTPConnectMethod()	R	R
HTTPWriteHeaders()	R	R
HTTPReadHeaders()	R	R
HTTPReadBody()	R	R
HTTPWriteBody()	R	R

[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

CloudTransport Action(s)[a,e]	HTTP HEAD and DELETE	HTTP GET	HTTP POST and PUT
HTTPConnectMethod()	1[b]	1	1
HTTPWriteHeaders()	1	1	1
HTTPWriteBody()	0	0	[1 - N]
HTTPReadHeaders()	1[c]	1[c]	1
HTTPReadBody()	0	[0 to N][d]	[0 to N][d]

[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 incidates 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()*

Argument	Direction	relatedStateVariable
Host	*IN*	*A_ARG_TYPE_Host*
MethodLine	*IN*	*A_ARG_TYPE_Headers*
Identifier	*OUT*	*A_ARG_TYPE_Identifier*
ConnectWriteCount	*OUT*	*A_ARG_TYPE_UI4*

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 toon 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 distinquish 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 - *HTTPConnectMehod()*, *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()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
700	Not Connected	The connection failed.
701	Host not found	The indicated host could not be found
702	Invalid Connection	The connection indicated in *Identifier* is not recognized
703	Method in Queue	A METHOD LINE was already in the HTTP REQUEST buffer, flush or resend.
704	HTTPS not allowed	The control point is not allowed an HTTPS connection.

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()*

Argument	Direction	relatedStateVariable
Identifier	*IN*	*A_ARG_TYPE_Identifier*
Headers	*IN*	*A_ARG_TYPE_Headers*
ConnectWriteCount	*OUT*	*A_ARG_TYPE_UI4*

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]
DATE: Mon, 12 Oct 2015 13:18:19 GMT<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>"
```

[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.

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.

6.6.3.4 Argument ConnectWriteCount

This output argument indicates the number of CloudTransport write related actions - HTTPConnectMehod(), 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 6-8 — Error Codes for HTTPWriteHeaders()

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

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()

Argument	Direction	relatedStateVariable

Argument	Direction	relatedStateVariable
Identifier	IN	A_ARG_TYPE_Identifier
CRLFFlag	IN	A_ARG_TYPE_Flag
Headers	OUT	A_ARG_TYPE_Headers
ReadLength	OUT	A_ARG_TYPE_UI4
ConnectReadCount	OUT	A_ARG_TYPE_UI4

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 nd 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=0170000000000000
0000000000000000<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.

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".

[1] Note that some line formatting has been added to improve readability.

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 6-10 — Error Codes for *HTTPReadHeaders()*

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

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()*

Argument	Direction	relatedStateVariable
Identifier	*IN*	*A_ARG_TYPE_Identifier*
Size	*IN*	*A_ARG_TYPE_UI4*
CRLFFlag	*IN*	*A_ARG_TYPE_Flag*
Body	*OUT*	*A_ARG_TYPE_Body*
ReadLength	*OUT*	*A_ARG_TYPE_UI4*
ConnectReadCount	*OUT*	*A_ARG_TYPE_UI4*

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 invocations[1]) to retrieve the entire body.

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

```
"a709ae369b6114a345e09c3e09d004fffb9260cc8005715f57d3067b6a700afae9316
66d0d754b5d0798cda129a26cb4b28974f9bdb8818486a310f7424da4a71fbb2863ec5
5b27e7b21345fffff6ae9a6dfdb4aab6dbcc64e9a6bbaa8368b8090fb2a7923b33cf19
3eba18b25cb0024c6926e5ffdda102e45d4bc909bb2a62329ce565f1325dd1d279b119
3c5901d54f289213d8d983a471a95174d0a867fa534abe3b259890a4af37fef9d"
```

to its BASE64 equivalent:

```
"YTcwOWFlMzY5YjYxMTRhMzQ1ZTA5YzNlMDlkMDA0ZmZmYjkyNjBjYzgwMDU3MTVmNTdkMz
A2N2I2YTcwMGFmYWU5MzE2NjZkMGQ3NTRiNWQwNzk4Y2RhMTI5YTI2Y2I0YjI4OTc0Zjli
ZGI4ODE4NDg2YTMxMGY3NDI0ZGE0YTcxZmJiMjg2M2VjNTViMjdlN2IyMTM0NWZmZmZmNm
FlOWE2ZGZkYjRhYWI2ZGJjYzY0ZTlhNmJiYWE4MzY4YjgwOTBmYjJhNzkyM2IzM2NmMTkz
ZWJhMThiMjVjYjAwMjRjNjkyNmU1ZmZkZGExMDJlNDVkNGJjOTA5YmIyYTYyMzI5Y2U1Nj
```

[1] 4001377 ÷ 6144 = 651.267

VmMTMyNWRkMWQyNzliMTE5M2M1OTAxZDU0ZjI4OTIxM2Q4ZDk4M2E0NzFhOTUxNzRkMGE4
NjdmYTUzNGFiZTNiMjU5ODkwYTRhZjM3ZmVmOWQ="

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 6-12 — Error Codes for *HTTPReadBody()*

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

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()*

Argument	Direction	relatedStateVariable
Identifier	*IN*	*A_ARG_TYPE_Identifier*
Body	*IN*	*A_ARG_TYPE_Body*
Size	*IN*	*A_ARG_TYPE_UI4*
ConnectWriteCount	*OUT*	*A_ARG_TYPE_UI4*

6.6.6.2 Argument *Identifier*

This input argument provides a temporary, unique identifier that allows the CloudTransport service to distinquish 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 - *HTTPConnectMehod()*, *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()*

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

7 THEORY OF OPERATIONS (INFORMATIVE)

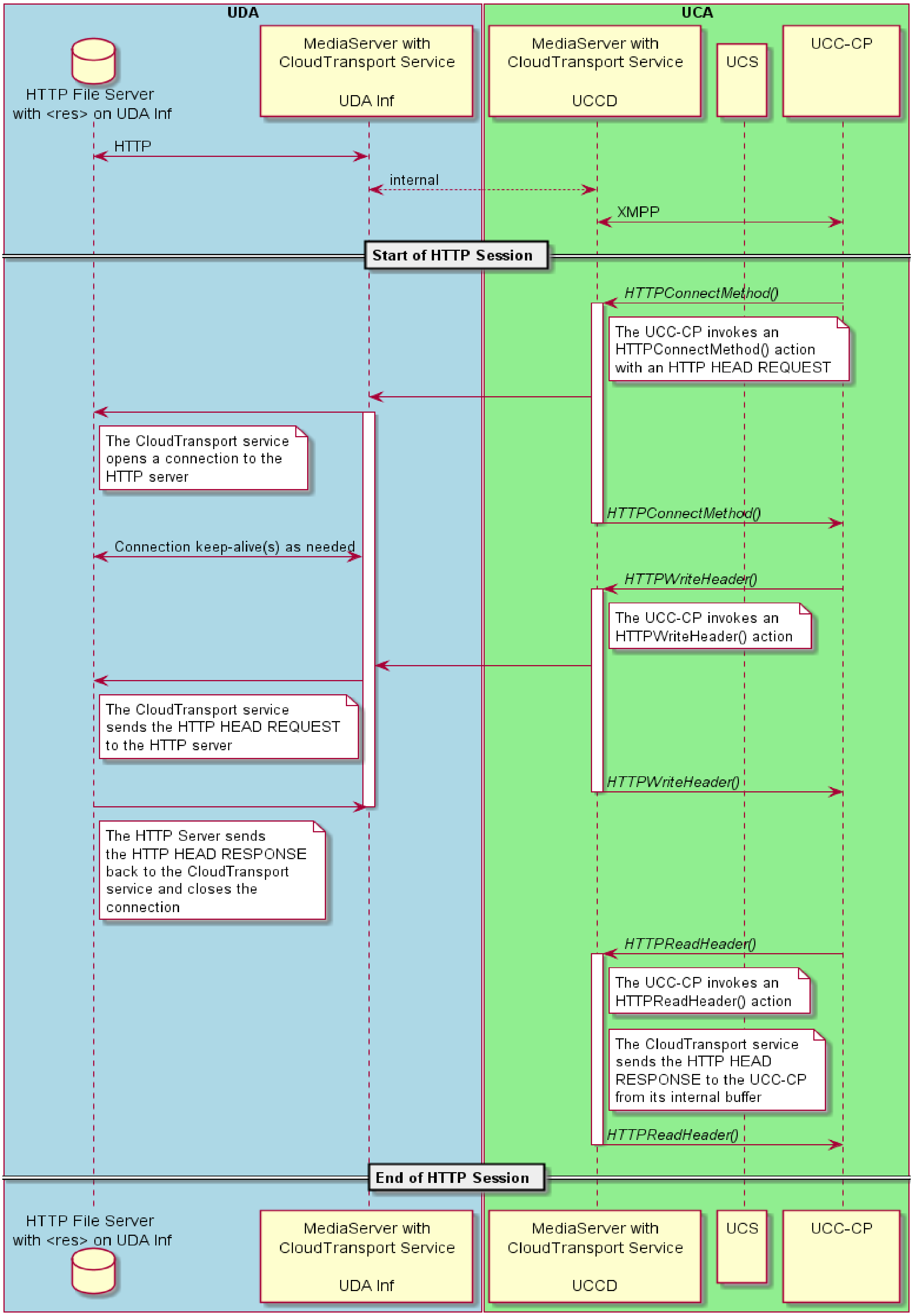
This section describes several typical scenarios encountered where the CloudTransport service is utilized. The device supporting the CloudTransport service will provide socket and TCP support, including TCP keep-alives on its local interface where the HTTP Server and Client will run. It will also need 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


UDA
UCA
HTTP File Server with <res> on UDA Inf
MediaServer with CloudTransport Service UDA Inf
MediaServer with CloudTransport Service UCCD
UCS
UCC-CP
HTTP
internal
XMPP
Start of HTTP Session
HTTPConnectMethod()
The UCC-CP invokes an HTTPConnectMethod() action with an HTTP HEAD REQUEST
The CloudTransport service opens a connection to the HTTP server
HTTPConnectMethod()
Connection keep-alive(s) as needed
HTTPWriteHeader()
The UCC-CP invokes an HTTPWriteHeader() action
The CloudTransport service sends the HTTP HEAD REQUEST to the HTTP server
HTTPWriteHeader()
The HTTP Server sends the HTTP HEAD RESPONSE back to the CloudTransport service and closes the connection
HTTPReadHeader()
The UCC-CP invokes an HTTPReadHeader() action
The CloudTransport service sends the HTTP HEAD RESPONSE to the UCC-CP from its internal buffer
HTTPReadHeader()
End of HTTP Session

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(
   "MxMGY3NDI0ZGE0YTcxZmJiMjg2M2VjNTViMjdlN2IyMTM",
   "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(
   MxMGY3NDI0ZGE0YTcxZmJiMjg2M2VjNTViMjdlN2IyMTM,
   HOST: 192.168.0.5:37222<CR><LF>
    DATE: Mon, 12 Oct 2015 13:18:19 GMT<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 CloudTranport 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 HTTPReadHeaders()

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

```
Request (SOAP/XMPP):
HTTPReadHeaders(
   MxMGY3NDI0ZGE0YTcxZmJiMjg2M2VjNTViMjdlN2IyMTM,
   0
)
```

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:

```
Response (SOAP/XMPP)
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
)
```

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 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 HTTPConnectMethod() followed by HTTPWriteHeaders(), HTTPReadHeaders() and then a series of 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

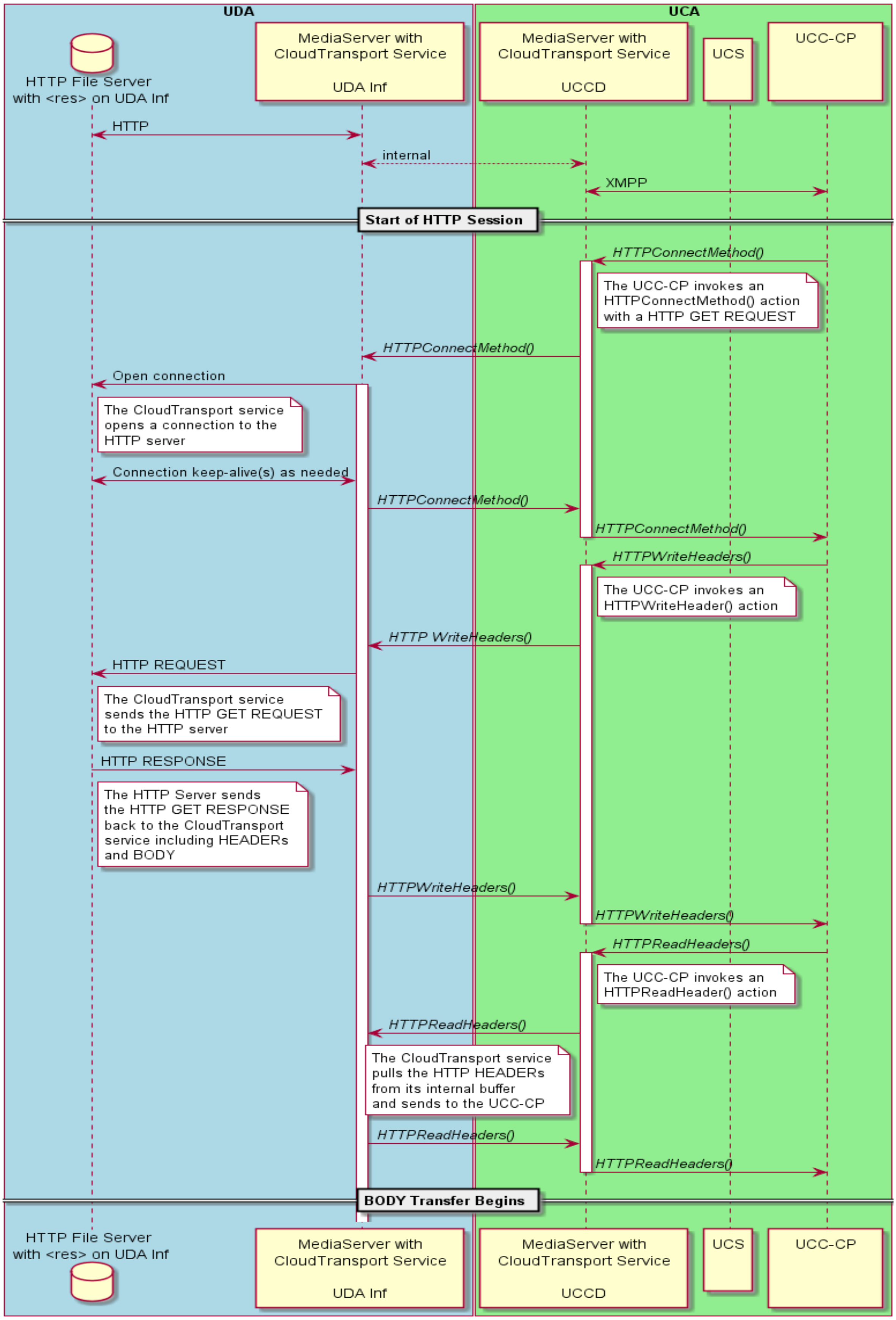
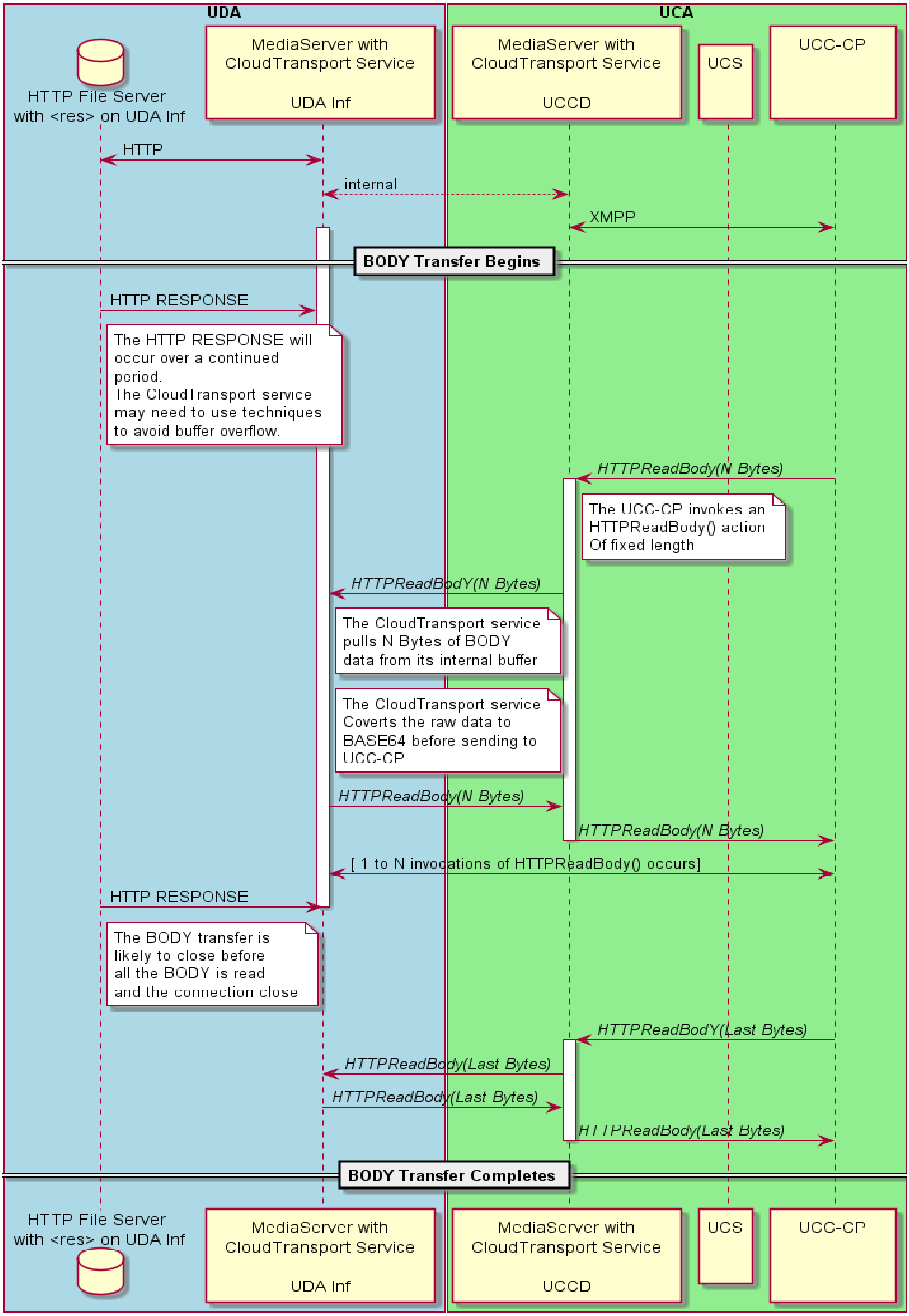

UDA
UCA
HTTP File Server with <res> on UDA Inf
MediaServer with CloudTransport Service UDA Inf
MediaServer with CloudTransport Service UCCD
UCS
UCC-CP
HTTP
internal
XMPP
Start of HTTP Session
HTTPConnectMethod()
The UCC-CP invokes an HTTPConnectMethod() action with a HTTP GET REQUEST
HTTPConnectMethod()
Open connection
The CloudTransport service opens a connection to the HTTP server
Connection keep-alive(s) as needed
HTTPConnectMethod()
HTTPConnectMethod()
HTTPWriteHeaders()
The UCC-CP invokes an HTTPWriteHeader() action
HTTP WriteHeaders()
HTTP REQUEST
The CloudTransport service sends the HTTP GET REQUEST to the HTTP server
HTTP RESPONSE
The HTTP Server sends the HTTP GET RESPONSE back to the CloudTransport service including HEADERs and BODY
HTTPWriteHeaders()
HTTPWriteHeaders()
HTTPReadHeaders()
The UCC-CP invokes an HTTPReadHeader() action
HTTPReadHeaders()
The CloudTransport service pulls the HTTP HEADERs from its internal buffer and sends to the UCC-CP
HTTPReadHeaders()
HTTPReadHeaders()
BODY Transfer Begins

Figure 7-3 HTTP GET REQUEST and RESPONSE Part 2



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.7.1.1.3. 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),

```
0023565c61690023542e7ca90800450005dccd4540004006e66ac0a80005c0a8001684
aafc0fc73ef2ceb09cca14501000edd3c60000494433030000000020f76415049430000
1f48000000696d6167652f6a7067000000ffd8ffe000104a46494600010100000010
 . . .
b48d233065307ed3e77d3ef6cfc315b709f19c9aabc72c97df607b62e1d018e58a4588
7ca411870cfdc7392690f5ee5487e11ddc1a25ee9ababdb11776f6b13486062cad0b67
239e87d3b1aded4bc0b75a97f664c750823b9d36e966b76f2cb829cef57cf2c5b
```

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

Response (SOAP/XMPP):

```
HTTPReadBody(
   MDAyMzU2NWM2MTY5MDAyMzU0MmU3Y2E5MDgwMDQ1MDAwNWRjY2Q0NTQwMDA0MDA2ZTY2YWMw
   YTgwMDA1YzBhODAwMTY4NGFhZmMwZmM3M2VmMmNlYjA5Y2NhMTQ1MDEwMDBlZGQzYzYwMDAw
   NDk0NDMzMDMwMDAwMDAwMjBmNzY0MTUwNDk0MzAwMDAxZjQ4MDAwMDAwNjk2ZDYxNjc2NTJm
   N…
   GM3MzkyNjkwZjVlZTU0ODdlMTFkZGMxYTI1ZWU5YWJhYmRiMTE3NzZmNmIxMzQ4NjA2MmNhZ
   DBiNjcyMzllODdkM2IxYWRlZDRiYzBiNzVhOTdmNjY0Yzc1MDgyM2I5ZDM2ZTk2NmI3NmYyY
   2I4MjljZWY1N2NmMmM1Yg==,
   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(
   2VjNTViMjdlN2MxMGY3NDI0ZGIyMTME0YTcxZmJiMjg2M,
   5377,
   0
)
```

with response

Response (SOAP/XMPP):

```
HTTPReadBody(
   . . .
   AwMDAwNDg2OTczNzQ2ZjcyNzkzYTIwNDE2ZDY1NzI2OTYzNjEyNzczMjA0NzcyNjU2MTc0Nj
   U3Mzc0MjA0ODY5MzEzOTM3MzUyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMjAyMDIwMj
   AyMDIwMjAyMDIwMjAyMDIwMjAyMDAwMDEwYw==,
   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> "00D0A" 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(
   MjdlN2MxMGY3NDI0ZGIyMT2VjNTVi,
   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(
   MjdlN2MxMGY3NDI0ZGIyMT2VjNTVi,
   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 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>
                    <direction>in</direction>
                    <relatedStateVariable>
                        A_ARG_TYPE_Headers
                    </relatedStateVariable>
                </argument>
                <argument>
                    <name>Identifier</name>
                    <direction>out</direction>
                    <relatedStateVariable>
                        A_ARG_TYPE_Identifier
                    </relatedStateVariable>
                </argument>
                <argument>
                    <name>ConnectWriteCout</name>
                    <direction>out</direction>
                    <relatedStateVariable>
                        A_ARG_TYPE_UI4
                    </relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <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>
            <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>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>
      </action>
   </actionList>
   <serviceStateTable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Host</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Headers</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Identifier</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Body</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_UI4</name>
         <dataType>ui4</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Flag</name>
         <dataType>boolean</dataType>
      </stateVariable>
   </serviceStateTable>
</scpd>
```