MOST

Media **O**riented **S**ystems **T**ransport

Multimedia and Control Networking Technology

Application Note UPnP over MOST
Rev. 1.0
06/2015

MOSTCO CONFIDENTIAL

See page 3 for the terms of disclosure

Legal Notice

COPYRIGHT

LICENSE DISCLAIMER

Nothing on any MOST Cooperation Web Site, or in any MOST Cooperation document, shall be construed as conferring any license under any of the MOST Cooperation or its members or any third party's intellectual property rights, whether by estoppel, implication, or otherwise.

CONTENT AND LIABILITY DISCLAIMER

MOST Cooperation or its members shall not be responsible for any errors or omissions contained at any MOST Cooperation Web Site, or in any MOST Cooperation document, and reserves the right to make changes without notice. Accordingly, all MOST Cooperation and third party information is provided "AS IS". In addition, MOST Cooperation or its members are not responsible for the content of any other Web Site linked to any MOST Cooperation Web Site. Links are provided as Internet navigation tools only.

MOST COOPERATION AND ITS MEMBERS DISCLAIM ALL WARRANTIES WITH REGARD TO THE INFORMATION (INCLUDING ANY SOFTWARE) PROVIDED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

In no event shall MOST Cooperation or its members be liable for any damages whatsoever, and in particular MOST Cooperation or its members shall not be liable for special, indirect, consequential, or incidental damages, or damages for lost profits, loss of revenue, or loss of use, arising out of or related to any MOST Cooperation Web Site, any MOST Cooperation document, or the information contained in it, whether such damages arise in contract, negligence, tort, under statute, in equity, at law or otherwise.

FEEDBACK INFORMATION

Any information provided to MOST Cooperation in connection with any MOST Cooperation Web Site, or any MOST Cooperation document, shall be provided by the submitter and received by MOST Cooperation on a non-confidential basis. MOST Cooperation shall be free to use such information on an unrestricted basis.

TRADEMARKS

MOST Cooperation and its members prohibit the unauthorized use of any of their trademarks. MOST Cooperation specifically prohibits the use of the MOST Cooperation LOGO unless the use is approved by the Steering Committee of MOST Cooperation.

SUPPORT AND FURTHER INFORMATION

For more information on the MOST technology, please contact:

MOST Cooperation
Administration
Emmy-Noether-Str. 14
76131 Karlsruhe
Germany

Tel: (+49) (0) 721 966 50 00

E-mail: contact@mostcooperation.com
Web: www.mostcooperation.com

Contents

Bibliography

All documents, which are referenced by this MOST document, are listed here along with their versions.

Document		Revision
[1]	MOST Specification	3.0
[2]	MOST FBlock template GeneralFBlock	3.0.6
[3]	“UPnP Device Architecture version 1.0” October 15, 2008.	1.0
[4]	Intel.com, “Open Software Projects”, http://opentools.homeip.net/dev-tools-for-upnp.	-
[5]	Reiter, S et al.: UPnP Enables Universal Control of In-Vehicle MOST Devices. Special Edition March 2014, p. 31-33.	-

Table Bibliography-1: Document references

Document History

Revision 1.0

Change Ref.	Section	Changes
1V0_001		Initial version.

1 Overview

This application note “UPnP over MOST” shows how to use the UPnP standard in a MOST based environment. UPnP is used to control potential in-vehicle devices, to foster the extension of consumer devices with controlling capability currently known from integrated head units. It is shown how the UPnP ProtocolInfo can be extended to cover MOST specific addressing, especially to support the use of MOST synchronous connections.

2 Introduction

The UPnP communication is realized within MOST by using the MOST Ethernet Protocol (MEP). In Figure 1 a sketch of the UPnP/AV scenario is shown. It consists of a UPnP control point, an audio source, called media server, and an audio sink, called media renderer, which are connected by the MOST bus. Each UPnP device implements a set of services, the media server e.g. a content directory service and a connection manager, which are used for device control. The UPnP control point is used to control both the media server and the media renderer using the according UPnP services. The source provides the audio data and sends it over the MOST synchronous connection to the associated audio sink.

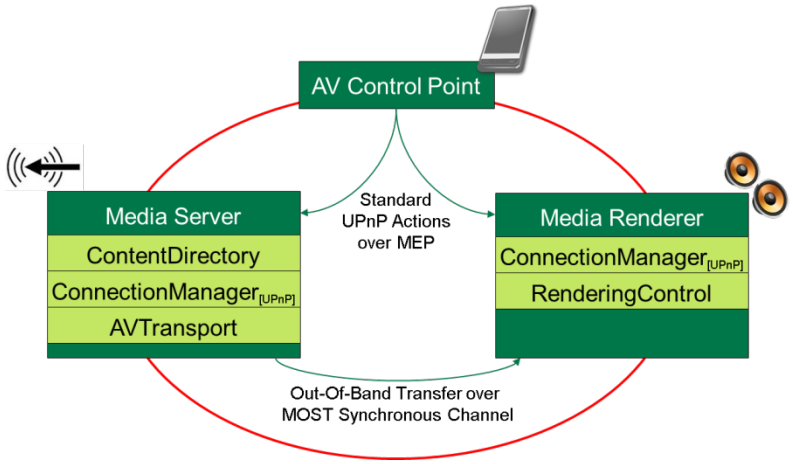

Figure 1: Structure of the UPnP/AV Scenario

3 The UPnP Standard

The goal of Universal Plug and Play (UPnP) is to enable a manufacturer-independent control of networked devices. The UPnP standard distinguishes two general device classes: controlled devices and controlling devices. The controlled device provides functionality that is managed by a controlling device or control point. UPnP defines protocols for the communication between a control point and the controlled devices. These protocols handle device discovery, description, and controlling with the help of internet protocols such as IP, TCP, UDP, and HTTP. Every transport channel that supports IP-based communication is suited. To control a variety of devices across manufacturer domains, the UPnP protocol specifies a set of Standardized Device Control Protocols (SDCP).

3.1 Standardized Device Control Protocols

An SDCP is a set of standardized interfaces, which have to be provided on each UPnP device in order to allow generic controlling of the latter. The SDCPs are grouped by their functionality such as A/V systems, with media server and media renderer or HVAC (Heating, Ventilation and Air Conditioning) systems. In the following, a few SDCPs of potential in-vehicle devices are listed. All SDCPs are standardized and published by the UPnP forum.

- *Basic_Device::BasicService*
  Provides no controlling functionality, used only to discover the device via UPnP
- *DeviceManagement ::ConfigurationManagement*
  Allows for managing the configuration in terms of parameters supported by the device
- *HVAC::HVAC_FanOperatingMode and HVAC::ControlValve*
  Different services can be used to control a HVAC system
- *LightingControls::DimmableLight*
  Turn a light source on or off. Changing the intensity of the light source in intermediate steps

These UPnP services are used to control a single device. In the following, a more complex scenario is regarded. In an audio/video scenario, two devices and the data transfer in between have to be controlled. In an A/V system, the following SDCPs have to be provided (see Figure 1).

- *MediaServerMediaRenderer::ConnectionManager*
  Perform capability matching between source/server devices and sink/renderer devices
  Find information about ongoing transfers in the network
  Set up and tear down connections between devices
- MediaServer::ContentDirectory
  Enable browsing the objects stored on the media server such as songs, photos, videos or broadcast program
- MediaRenderer::RenderingControl
  Control how the current content is rendered, e.g., brightness, contrast, volume, balance or equalizer settings
- MediaServerMediaRenderer::AVTransport
  Enable control over the transport of audio and video streams

In the following example, the connection manager service is regarded more closely.

3.1.1 Example: UPnP-ConnectionManager

The minimal requirements to implement the connection manager service are presented in this section. The UPnP-ConnectionManager service is used for retrieving connection information or for establishing/allocating a connection. It is the main service that is used for creating a connection utilizing a MOST synchronous connection. To implement this service the standard specifies a set of state variables and functions that have to be provided. Further variables and functions are declared optional. The necessary state variables are shown in Table 2 and the required actions are highlighted in Table 3.

Variable Name	R/O	Data Type	Allowed Value
SourceProtocolInfo	R	string	CSV (string)
SinkProtocolInfo	R	string	CSV (string)
CurrentConnectionIDs	R	string	CSV (ui4)
FeatureList	R	string	Features XML Document

Table 2: Required state variables

The state variables contain the source/sink protocol info which specifies the capabilities of the device, for example, which media format can be sent or received (see end of this section). CurrentConnectionIDs is a list of all currently managed connections. FeatureList contains the features of the device, such as synchronized playback.

Action Name	R/O	Parameter		
GetProtocolInfo()	R	Source	OUT	SourceProtocolInfo
		Sink	OUT	SinkProtocolInfo
PrepareForConnection()	O	Indicate whether or not it can establish a connection; used if multiple connections are possible at the same time		
		RemoteProtocolInfo	IN	A_ARG_TYPE_ProtocolInfo
		PeerConnectionManager	IN	A_ARG_TYPE_ConnectionManager
		PeerConnectionID	IN	A_ARG_TYPE_ConnectionID
		ConnectionID	OUT	A_ARG_TYPE_ConnectionID
		AVTransportID	OUT	A_ARG_TYPE_AVTransportID
		RcsID	OUT	A_ARG_TYPE_RcsID
GetCurrentConnectionIDs()	R	Comma-Separated Value list of ConnectionIDs		
GetCurrentConnectionInfo()	R	Returns associated information of the connection referred to by the ConnectionID		
ConnectionComplete()	O			
GetFeatureList()	R			

Table 3: Required and optional actions

At least the functions in Table 3 marked with 'R' have to be provided for each connection manager. They contain getter functions for the Sink/SourceProtocolInfo, FeatureList, or the CurrentConnectionIDs. GetCurrentConnectionInfo() can be used to get a detailed description of the current connection. This function will be used to forward the connection label in the MOST synchronous scenario. The optional functions PrepareForConnection() and ConnectionComplete() are used to establish/allocate a connection or internal resource.

3.1.1.1 ProtocolInfo

The UPnP-ConnectionManager service defines the notion of ProtocolInfo as information required by a control point in order to determine (a certain level of) compatibility between the streaming mechanisms. A ProtocolInfo entry is specified as string and formatted as follows:

<protocol>':'<network>':'<contentFormat>':'<additionalInfo>

Each of the four elements may be a wildcard '*'. The matching between devices is performed by string comparison operations. The ProtocolInfo string consists of a <protocol>, <network>, <contentFormat>, and <additionalInfo> field, where the <protocol> entry specifies the transfer protocol, e.g. 'http-get'. The <network> field specifies the used network, that is, in the case of TCP/IP, mostly a wildcard '*', because the devices belong to the same IP network, based on the UPnP device discovery. The <contentFormat> describes the exchanged content format such as 'audio/mpeg'. The <additionalInfo> field conveys any additional information. Table 4 shows the general structure of the protocol information string and an example of an http-get specification. The <additionalInfo> part does not need to match between source and sink. Its purpose is to convey any additional information needed to set up the out of band stream. The structure of the <additionalInfo> part is a list of values (separated by a semi-colon) that has the following format:

<org-name>_<token-name>=<value>

The <org-name> is the ICANN-registered domain name, e.g., 'upnp.org'. The <token-name> consists of an alpha-numeric character sequence. There exists a set of specified protocols.

To establish a connection between two devices, the control point uses the GetProtocol() function to retrieve information about the connection types a device supports either as sink or as source. To find a pair that can communicate, the provided source protocols have to match the required sink protocols.

The control point only compares the provided strings; to be more precise: the different strings of the comma-separated list. Wildcards indicate that this field matches any entry. When a suitable pair of devices is found, the control point can allocate a connection between these two devices. To this end, the function PrepareForConnection() is used. It allocates a communication channel or resource within the device. If the device does not provide such a function, a standard connection already exists and no previous resource allocation is necessary. For example, in the case of an HTTP data exchange no resources have to be allocated in advance as the TCP/IP connection is established when sending the http-get request. Therefore, this function is optional. Similar to PrepareForConnection(), the function ConnectionComplete() frees the previously allocated resource(s). If one of the functions shall be used, both functions have to be implemented. The control point can get information about the current allocated connections with the functions GetCurrentConnectionIDs() and GetCurrentConnectionInfo().

3.2 Device Discovery

The UPnP protocol handles device discovery. The interaction between control point und controllable devices can be separated into 5 steps. In a first step, UPnP devices are detected. In the second step, the control point detects the device capabilities. In step 3, the control point uses the SDCPs to interact with the device. During communication the control point can listen to state changes in the UPnP devices, called eventing (step 4).

After a controllable UPnP device is associated with an IP address, it starts advertising its services. On the other hand, when a control point enters a network it sends a search command to find UPnP devices within the network. Advertise and search messages are based on the Simple Service Discovery Protocol (SSDP) and on multicast messages. The advertisements or the search responses contain an URL for more information about the device, which is stored in a XML file. This root-description can be downloaded by the control point with a separate http-get request. The root-description contains an overview of the services provided by the device and an URL to download a more detailed description of the provided service. After the receipt of the root-description the control point starts downloading the service description with separate http-get requests. After receipt of all service descriptions, the control point enters step 3, controlling the device.

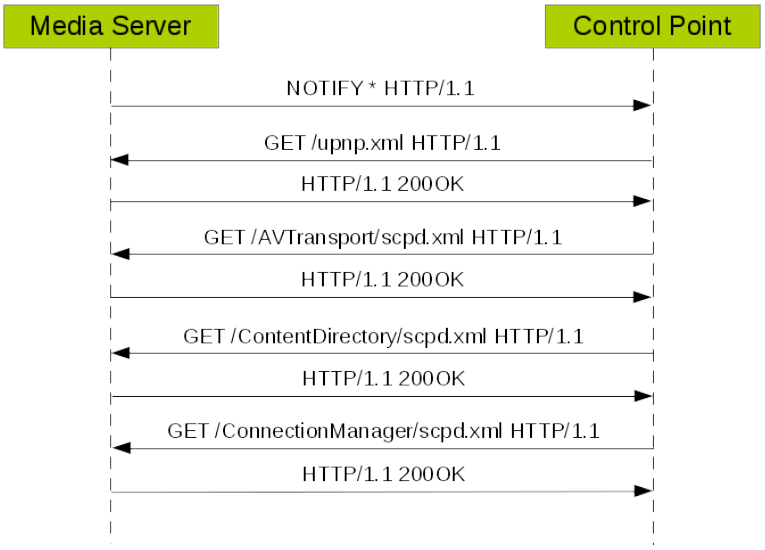

Figure 2: Device discovery sequence in the AV-scenario

4 Adaption to the UPnP Standard

In order to be able to use the UPnP standard to control the MOST synchronous data transmission, different extensions are necessary. The UPnP standard specifies the protocol information string for different network types and media content. MOST is not natively supported by the standard. But there are rules how vendor extensions can be made to the standard. In the first section the proposed protocol information format is presented. In the next sections, it is shown how the standardized action interfaces can be used to establish and transmit data via a MOST synchronous connection.

4.1 ProtocolInfo Extension

UPnP standardizes the interfaces to control and manage the devices, but it does not describe anything related to the out-of-band, non-UPnP transfer protocol, for transmission of the actual content. The concept of the ProtocolInfo is used to determine compatibility between the devices and initiate transfers. The ProtocolInfo is exchanged between media server and renderer to identify the device capabilities or to establish the out-of-band connection. The control point uses only string operations to handle the ProtocolInfo to find compatible devices. The control point does not use any semantic information. Exclusively, the media server and media renderer interpret the exchanged string, allowing standard control points to handle a vendor specific ProtocolInfo, for instance, the MOST extensions, see section 3.1.1.1.

	<protocol>	<network>	<contentFormat>	<additionalInfo>
HTTP	“http-get”	“*”	“audio/mpeg”	“*”
MOST	**ICANN Domain Name**	**SourceInfo:: TransmissionClass**	**SourceInfo:: ContentType**	**SourceInfo::***
	“mostcooperation.com”	“Synchronous”	“Audio”	“mostcooperation.com_DataConnectionLabel=66”
		“PacketizedIsochronous”	“SPDIF”	“mostcooperation.com_AudioChannels=6”
		“Asynchronous”	“GenericPCM”	
		...	**...**	**...**

Table 4: Example of the MOST extensions for the ProtocolInfo

MOST protocol information is not part of the UPnP standard, but the standard provides the ability to realize a vendor-specific ProtocolInfo structure. For the MOST extension, a structure based on the SourceInfo (0x100) function of the GeneralFBlock is proposed. For vendor extensions, the <protocol> field should contain the ICANN domain name of the vendor: ‘mostcooperation.com’.

The <network> field contains the SourceInfo::TransmissionClass information of the GeneralFBlock [2]. The <contentFormat> field is filled with the SourceInfo::ContentType information. The <additionalInfo> field content is based on the <network> and <contentFormat> fields. The <contentFormat> ‘GenericPCM’ requires different additional information than the <contentFormat> ‘Audio’. The UPnP standard allows a semicolon-separated list for specifying multiple entries. A single entry has the following structure:

‘<org-name>_<token_name>=<value>’.

The ‘Synchronous’ network, for example, specifies the ‘DataConnectionLabel’ in the <additionalInfo> field. The entry for a connection label 0x43 would be: ‘mostcooperation.com_DataConnectionLabel=66’.

4.2 Push Architecture

The most suited architecture for using MOST synchronous connections is the push architecture. In the push architecture, the content provider controls the transmission of the content data, for example, it starts/stops data streaming. Taking the MOST synchronous scenario, the content provider starts streaming the audio content. When finished, the content provider, that is, the media server, stops the media streaming. In this case, the responsibility to allocate a synchronous channel is with the media server. The challenge for the UPnP control point is to forward the connection label from the media server to the media renderer.

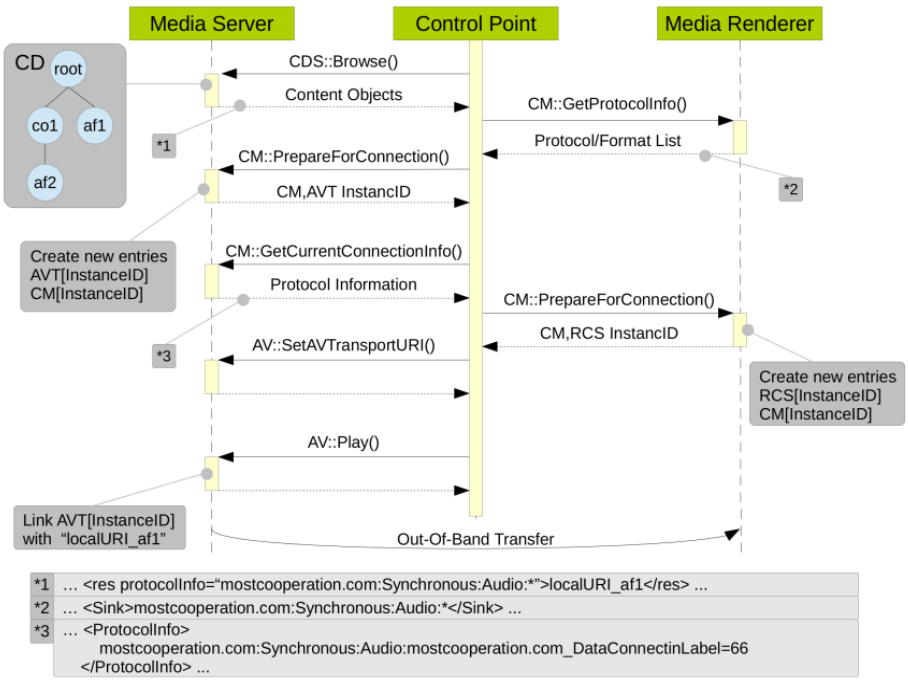

Figure 3: Connection establishment in a push architecture

In Figure 3, the sequence to establish a connection between the media server and the media renderer is shown. First, the control point calls the media server and browses the content provided. Media content is represented by a tree structure with containers (container co1) and the actual content (audio file af1, af2). The browse function is used to traverse the tree and display meta-information of the different entries. Filters can be used to address dedicated information. Each content object, such as an audio file, contains a URI, which is used to identify the content within the media server. Additionally, it contains a ProtocolInfo entry which defines how the content can be transferred; see *1 in Figure 3, which specifies that for the object with the URI 'localURI_af1', a MOST synchronous connection has to be used. The control point now checks the media renderer whether the required ProtocolInfo is supported (*2 in Figure 3). The control point only compares the ProtocolInfo of the media server with the information of the media renderer. This comparison is based on string matching with wildcards, so no semantic information is used by the control point. After a positive check, the control point prepares the data transfer. In the case of MOST synchronous connections, it is advisable to use the PrepareForConnection function. This allows media server and renderer to prepare for a new connection; in the case of the synchronous channel, the requested bandwidth is allocated. If this function is not provided, the media server and renderer have to use pre-allocated channels. The function is called with the desired ProtocolInfo and the requested direction of the connection. For outgoing connections on the media server, the PrepareForConnection function allocates the MOST synchronous connection and an internal connection (CM[InstanceID]) and AVTransport (AVT[InstanceID]) instance. These instances contain all relevant information regarding one connection. The return parameter is an instance ID that identifies this connection object within the device. Meaning the returned instance IDs are only unique within one device. For each subsequent call to the AVTransport or UPnP-ConnectionManager service, the instance ID is used to identify the allocated instance. This enables parallel managing of multiple streams. With the

GetProtocolConnectionInfo call, it is possible to retrieve detailed information about the connection. This call is used to get information such as the MOST data connection label, which is returned within the ProtocolInfo field (compare *3 in Figure 3). With this information, the PrepareForConnection function on the media renderer can be called. This time, an incoming connection shall be prepared, therefore the contained ProtocolInfo is used to connect to the previously allocated synchronous channel. After this, the out-of-band connection is established and the data transfer can be initialized. Therefore, the AVTransport service within the media server is called. The previous connection/AVTransport object is associated with the content information URI returned by the browse function. With the call SetAVTransportURI, the URI is set and a subsequent call of the function 'play' starts audio playback and streaming over the MOST synchronous connection.

The RenderingControl service of the media renderer provides functions for changing the volume.

4.3 Pull Architecture

In the pull architecture, the media renderer controls the data exchange. The pull architecture is not recommended for MOST, thus not described here.

Appendix A: UPnP Stack for the Reference Implementation (Informative)

Different UPnP stacks have been evaluated. The final choice for the reference implementation was the *Developer Tools for UPnP™ Technologies* from [3]. This is a set of development and reference tools for creating software that is compatible with the UPnP specifications. Especially, it is possible to generate source code for an UPnP stack which contains all UPnP service descriptions. Therefore, no file system has to be provided. Stub functions are created to handle the service requests.