INTERNATIONAL STANDARD

ISO/IEC 29341-24-10

First edition
2017-09

Information technology — UPnP Device Architecture —

Part 24-10: Internet gateway device control protocol — Level 2 — Wide area network internet protocol — Connection service

Technologies de l'information — Architecture de dispositif UPnP —

Partie 24-10: Protocole de contrôle de dispositif de passerelle Internet — Niveau 2 — Protocole internet de réseau étendu — Service de connexion

Reference number
ISO/IEC 29341-24-10:2017(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

CONTENTS

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see http://www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of Standard, the meaning of the ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword – Supplementary information

ISO/IEC 29341-24-10 was prepared by UPnP Forum and adopted, under the PAS procedure, by joint technical committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

The list of all currently available parts of ISO/IEC 29341 series, under the general title *Information technology — UPnP Device Architecture*, can be found on the ISO web site.

Introduction

ISO and IEC draw attention to the fact that it is claimed that compliance with this document may involve the use of patents as indicated below.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights. The holders of -these patent rights have assured ISO and IEC that they are willing to negotiate licenses under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC.

Intel Corporation has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Intel Corporation
Standards Licensing Department
5200 NE Elam Young Parkway
MS: JFS-98
USA – Hillsboro, Oregon 97124

Microsoft Corporation has informed IEC and ISO that it has patent applications or granted patents as listed below:

6101499 / US; 6687755 / US; 6910068 / US; 7130895 / US; 6725281 / US; 7089307 / US; 7069312 / US; 10/783 524 /US

Information may be obtained from:

Microsoft Corporation
One Microsoft Way
USA – Redmond WA 98052

Philips International B.V. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Philips International B.V. – IP&S
High Tech campus, building 44 3A21
NL – 5656 Eindhoven

NXP B.V. (NL) has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

NXP B.V. (NL)
High Tech campus 60
NL – 5656 AG Eindhoven

Matsushita Electric Industrial Co. Ltd. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Matsushita Electric Industrial Co. Ltd.
1-3-7 Shiromi, Chuoh-ku
JP – Osaka 540-6139

Hewlett Packard Company has informed IEC and ISO that it has patent applications or granted patents as listed below:

5 956 487 / US; 6 170 007 / US; 6 139 177 / US; 6 529 936 / US; 6 470 339 / US; 6 571 388 / US; 6 205 466 / US

Information may be obtained from:

Hewlett Packard Company
1501 Page Mill Road
USA – Palo Alto, CA 94304

Samsung Electronics Co. Ltd. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Digital Media Business, Samsung Electronics Co. Ltd.
416 Maetan-3 Dong, Yeongtang-Gu,
KR – Suwon City 443-742

Huawei Technologies Co., Ltd. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Huawei Technologies Co., Ltd.
Administration Building, Bantian Longgang District
Shenzhen – China 518129

Qualcomm Incorporated has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Qualcomm Incorporated
5775 Morehouse Drive
San Diego, CA – USA 92121

Telecom Italia S.p.A.has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Telecom Italia S.p.A.
Via Reiss Romoli, 274
Turin - Italy 10148

Cisco Systems informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA – USA 95134

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Original UPnP Document

Reference may be made in this document to original UPnP documents. These references are retained in order to maintain consistency between the specifications as published by ISO/IEC and by UPnP Implementers Corporation and later by UPnP Forum. The following table indicates the original UPnP document titles and the corresponding part of ISO/IEC 29341:

UPnP Document Title	ISO/IEC 29341 Part
UPnP Device Architecture 1.0	ISO/IEC 29341-1:2008
UPnP Device Architecture Version 1.0	ISO/IEC 29341-1:2011
UPnP Device Architecture 1.1	ISO/IEC 29341-1-1:2011
UPnP Device Architecture 2.0	ISO/IEC 29341-1-2
UPnP Basic:1 Device	ISO/IEC 29341-2
UPnP AV Architecture:1	ISO/IEC 29341-3-1:2008
UPnP AV Architecture:1	ISO/IEC 29341-3-1:2011
UPnP AVTransport:1 Service	ISO/IEC 29341-3-10
UPnP ConnectionManager:1 Service	ISO/IEC 29341-3-11
UPnP ContentDirectory:1 Service	ISO/IEC 29341-3-12
UPnP RenderingControl:1 Service	ISO/IEC 29341-3-13
UPnP MediaRenderer:1 Device	ISO/IEC 29341-3-2
UPnP MediaRenderer:2 Device	ISO/IEC 29341-3-2:2011
UPnP MediaServer:1 Device	ISO/IEC 29341-3-3
UPnP AVTransport:2 Service	ISO/IEC 29341-4-10:2008
UPnP AVTransport:2 Service	ISO/IEC 29341-4-10:2011
UPnP ConnectionManager:2 Service	ISO/IEC 29341-4-11:2008
UPnP ConnectionManager:2 Service	ISO/IEC 29341-4-11:2011
UPnP ContentDirectory:2 Service	ISO/IEC 29341-4-12
UPnP RenderingControl:2 Service	ISO/IEC 29341-4-13:2008
UPnP RenderingControl:2 Service	ISO/IEC 29341-4-13:2011
UPnP ScheduledRecording:1	ISO/IEC 29341-4-14
UPnP ScheduledRecording:2	ISO/IEC 29341-4-14:2011
UPnP MediaRenderer:2 Device	ISO/IEC 29341-4-2
UPnP MediaServer:2 Device	ISO/IEC 29341-4-3
UPnP AV Datastructure Template:1	ISO/IEC 29341-4-4:2008
UPnP AV Datastructure Template:1	ISO/IEC 29341-4-4:2011
UPnP DigitalSecurityCamera:1 Device	ISO/IEC 29341-5-1
UPnP DigitalSecurityCameraMotionImage:1 Service	ISO/IEC 29341-5-10
UPnP DigitalSecurityCameraSettings:1 Service	ISO/IEC 29341-5-11
UPnP DigitalSecurityCameraStillImage:1 Service	ISO/IEC 29341-5-12
UPnP HVAC_System:1 Device	ISO/IEC 29341-6-1
UPnP ControlValve:1 Service	ISO/IEC 29341-6-10
UPnP HVAC_FanOperatingMode:1 Service	ISO/IEC 29341-6-11
UPnP FanSpeed:1 Service	ISO/IEC 29341-6-12
UPnP HouseStatus:1 Service	ISO/IEC 29341-6-13
UPnP HVAC_SetpointSchedule:1 Service	ISO/IEC 29341-6-14
UPnP TemperatureSensor:1 Service	ISO/IEC 29341-6-15
UPnP TemperatureSetpoint:1 Service	ISO/IEC 29341-6-16
UPnP HVAC_UserOperatingMode:1 Service	ISO/IEC 29341-6-17
UPnP HVAC_ZoneThermostat:1 Device	ISO/IEC 29341-6-2

UPnP BinaryLight:1 Device	ISO/IEC 29341-7-1
UPnP Dimming:1 Service	ISO/IEC 29341-7-10
UPnP SwitchPower:1 Service	ISO/IEC 29341-7-11
UPnP DimmableLight:1 Device	ISO/IEC 29341-7-2
UPnP InternetGatewayDevice:1 Device	ISO/IEC 29341-8-1
UPnP LANHostConfigManagement:1 Service	ISO/IEC 29341-8-10
UPnP Layer3Forwarding:1 Service	ISO/IEC 29341-8-11
UPnP LinkAuthentication:1 Service	ISO/IEC 29341-8-12
UPnP RadiusClient:1 Service	ISO/IEC 29341-8-13
UPnP WANCableLinkConfig:1 Service	ISO/IEC 29341-8-14
UPnP WANCommonInterfaceConfig:1 Service	ISO/IEC 29341-8-15
UPnP WANDSLLinkConfig:1 Service	ISO/IEC 29341-8-16
UPnP WANEthernetLinkConfig:1 Service	ISO/IEC 29341-8-17
UPnP WANIPConnection:1 Service	ISO/IEC 29341-8-18
UPnP WANPOTSLinkConfig:1 Service	ISO/IEC 29341-8-19
UPnP LANDevice:1 Device	ISO/IEC 29341-8-2
UPnP WANPPPConnection:1 Service	ISO/IEC 29341-8-20
UPnP WLANConfiguration:1 Service	ISO/IEC 29341-8-21
UPnP WANDevice:1 Device	ISO/IEC 29341-8-3
UPnP WANConnectionDevice:1 Device	ISO/IEC 29341-8-4
UPnP WLANAccessPointDevice:1 Device	ISO/IEC 29341-8-5
UPnP Printer:1 Device	ISO/IEC 29341-9-1
UPnP ExternalActivity:1 Service	ISO/IEC 29341-9-10
UPnP Feeder:1.0 Service	ISO/IEC 29341-9-11
UPnP PrintBasic:1 Service	ISO/IEC 29341-9-12
UPnP Scan:1 Service	ISO/IEC 29341-9-13
UPnP Scanner:1.0 Device	ISO/IEC 29341-9-2
UPnP QoS Architecture:1.0	ISO/IEC 29341-10-1
UPnP QosDevice:1 Service	ISO/IEC 29341-10-10
UPnP QosManager:1 Service	ISO/IEC 29341-10-11
UPnP QosPolicyHolder:1 Service	ISO/IEC 29341-10-12
UPnP QoS Architecture:2	ISO/IEC 29341-11-1
UPnP QosDevice:2 Service	ISO/IEC 29341-11-10
UPnP QosManager:2 Service	ISO/IEC 29341-11-11
UPnP QosPolicyHolder:2 Service	ISO/IEC 29341-11-12
UPnP QOS v2 Schema Files	ISO/IEC 29341-11-2
UPnP RemoteUIClientDevice:1 Device	ISO/IEC 29341-12-1
UPnP RemoteUIClient:1 Service	ISO/IEC 29341-12-10
UPnP RemoteUIServer:1 Service	ISO/IEC 29341-12-11
UPnP RemoteUIServerDevice:1 Device	ISO/IEC 29341-12-2
UPnP DeviceSecurity:1 Service	ISO/IEC 29341-13-10
UPnP SecurityConsole:1 Service	ISO/IEC 29341-13-11
UPnP ContentDirectory:3 Service	ISO/IEC 29341-14-12:2011
UPnP MediaServer:3 Device	ISO/IEC 29341-14-3:2011
UPnP ContentSync:1	ISO/IEC 29341-15-10:2011
UPnP Low Power Architecture:1	ISO/IEC 29341-16-1:2011
UPnP LowPowerProxy:1 Service	ISO/IEC 29341-16-10:2011

UPnP LowPowerDevice:1 Service	ISO/IEC 29341-16-11:2011
UPnP QoS Architecture:3	ISO/IEC 29341-17-1:2011
UPnP QosDevice:3 Service	ISO/IEC 29341-17-10:2011
UPnP QosManager:3 Service	ISO/IEC 29341-17-11:2011
UPnP QosPolicyHolder:3 Service	ISO/IEC 29341-17-12:2011
UPnP QosDevice:3 Addendum	ISO/IEC 29341-17-13:2011
UPnP RemoteAccessArchitecture:1	ISO/IEC 29341-18-1:2011
UPnP InboundConnectionConfig:1 Service	ISO/IEC 29341-18-10:2011
UPnP RADAConfig:1 Service	ISO/IEC 29341-18-11:2011
UPnP RADASync:1 Service	ISO/IEC 29341-18-12:2011
UPnP RATAConfig:1 Service	ISO/IEC 29341-18-13:2011
UPnP RAClient:1 Device	ISO/IEC 29341-18-2:2011
UPnP RAServer:1 Device	ISO/IEC 29341-18-3:2011
UPnP RADiscoveryAgent:1 Device	ISO/IEC 29341-18-4:2011
UPnP SolarProtectionBlind:1 Device	ISO/IEC 29341-19-1:2011
UPnP TwoWayMotionMotor:1 Service	ISO/IEC 29341-19-10:2011
UPnP AV Architecture:2	ISO/IEC 29341-20-1
UPnP AVTransport:3 Service	ISO/IEC 29341-20-10
UPnP ConnectionManager:3 Service	ISO/IEC 29341-20-11
UPnP ContentDirectory:4 Device	ISO/IEC 29341-20-12
UPnP RenderingControl:3 Service	ISO/IEC 29341-20-13
UPnP ScheduledRecording:2 Service	ISO/IEC 29341-20-14
UPnP MediaRenderer:3 Service	ISO/IEC 29341-20-2
UPnP MediaServer:4 Device	ISO/IEC 29341-20-3
UPnP AV Datastructure Template:1	ISO/IEC 29341-20-4
UPnP InternetGatewayDevice:2 Device	ISO/IEC 29341-24-1
UPnP WANIPConnection:2 Service	ISO/IEC 29341-24-10
UPnP WANIPv6FirewallControl:1 Service	ISO/IEC 29341-24-11
UPnP WANConnectionDevice:2 Service	ISO/IEC 29341-24-2
UPnP WANDevice:2 Device	ISO/IEC 29341-24-3
UPnP Telephony Architecture:2	ISO/IEC 29341-26-1
UPnP CallManagement:2 Service	ISO/IEC 29341-26-10
UPnP MediaManagement:2 Service	ISO/IEC 29341-26-11
UPnP Messaging:2 Service	ISO/IEC 29341-26-12
UPnP PhoneManagement:2 Service	ISO/IEC 29341-26-13
UPnP AddressBook:1 Service	ISO/IEC 29341-26-14
UPnP Calendar:1 Service	ISO/IEC 29341-26-15
UPnP Presense:1 Service	ISO/IEC 29341-26-16
UPnP TelephonyClient:2 Device	ISO/IEC 29341-26-2
UPnP TelephonyServer:2 Device	ISO/IEC 29341-26-3
UPnP Friendly Info Update:1 Service	ISO/IEC 29341-27-1
UPnP MultiScreen MultiScreen Architecture:1	ISO/IEC 29341-28-1
UPnP MultiScreen Application Management:1 Service	ISO/IEC 29341-28-10
UPnP MultiScreen Screen:1 Device	ISO/IEC 29341-28-2
UPnP MultiScreen Application Management:2 Service	ISO/IEC 29341-29-10
UPnP MultiScreen Screen:2 Device	ISO/IEC 29341-29-2
UPnP IoT Management and Control Architecture Overview:1	ISO/IEC 29341-30-1

UPnP DataStore:1 Service	ISO/IEC 29341-30-10
UPnP IoT Management and Control Data Model:1 Service	ISO/IEC 29341-30-11
UPnP IoT Management and Control Transport Generic:1 Service	ISO/IEC 29341-30-12
UPnP IoT Management and Control:1 Device	ISO/IEC 29341-30-2
UPnP Energy Management:1 Service	ISO/IEC 29341-31-1

1 Scope

This document specifies the characteristics of the UPnP networked service named *WANIPConnection*, version *2*. This service definition is compliant with *UPnP Device Architecture 1.0* [1] . It is one component of the Device Control Protocol for the UPnP Internet Gateway Device (see [3]).

This service enables a UPnP control point to:

- configure and control an IP connection between a LAN client on one side of an Internet gateway (see [3]) and a WAN host on the other side;
- manage any physical WAN interface—such as DSL or cable—that supports an IP connection.

This service is a required in a *WANConnectionDevice* (see [5]) when the device supports an IP connection.

2 Normative references

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

[1] *UPnP Device Architecture, version 1.0*, UPnP Forum, June 8, 2000.
Available at: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf.

[2] *UPnP Device Architecture, version 1.1*, UPnP Forum, October 15, 2008.
Available at: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf.

[3] *InternetGatewayDevice:2*, version 1.00, UPnP Forum, December 10, 2010.
Available at http://upnp.org/specs/gw/UPnP-gw-InternetGatewayDevice-v2-Device.pdf

[4] *WANDevice:2*, UPnP Forum, December 10, 2010
Available at http://upnp.org/specs/gw/UPnP-gw-WANDevice-v2-Device.pdf

[5] *WANConnectionDevice:2*, UPnP Forum, December 10, 2010
Available at http://upnp.org/specs/gw/UPnP-gw-WANIPConnection-v2-Service.pdf

[6] *WANIPConnection:1.* UPnP Forum, November 12, 2001
Available at http://upnp.org/specs/gw/UPnP-gw-WANIPConnection-v1-Service.pdf

[7] ISO 8601:2000, *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/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=26780&ICS1=1&ICS2=140&ICS3=30.

[8] IETF RFC 1035, *Domain names - implementation and specification*, P. Mockapetris, November 1987.
Available at: http://tools.ietf.org/html/rfc1035.

[9] IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter, January 2006.
Available at: http://tools.ietf.org/html/rfc3986.

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

[11] *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.

[12] *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.

3 Terms, definitions, symbols and abbreviated terms

For the purposes of this document, the terms and definitions given in [1], [2], [3] and the following apply.

3.1 Provision

3.1.1
A
allowed

3.1.2
conditionally allowed
CA
The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is allowed, otherwise it is not allowed.

3.1.3
conditionally required
CR
The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is required, otherwise it is not allowed.

3.1.4
not allowed
The definition or behavior is prohibited by this specification. Opposite of required.

3.1.5
R
required

3.1.6
X
vendor-defined, non-standard

3.2 Terms specific to network connections

3.2.1
client
a host located on the local network that connects to a remote host on the Internet through an Internet Gateway Device

3.2.2
IGD control point
UPnP control point that uses the UPnP IGD protocol to control an Internet Gateway Device

3.2.3
2-box model
configuration where the client and IGD control point are located in one physical device and the Internet Gateway Device is located in a second physical device

3.2.4
3-box model
configuration where the client, IGD control point and Internet Gateway Device are located in three separate physical devices.

3.2.5
port forwarding
port mapping rule which allows a remote host to reach a client port via a NAT enabled router

3.2.6
port triggering
method of automating port forwarding in which outbound traffic on predetermined ports causes inbound traffic to specific incoming ports to be dynamically forwarded to the initiating host, while the outbound ports are in use

3.3 Abbreviated terms

3.3.1
CSV
Comma Separated Value

3.3.2
IETF
Internet Engineering Task Force

3.3.3
IGD
Internet Gateway Device

3.3.4
MAC
Media Access Control

3.3.5
NAPT
Network Address Port Translation

3.3.6
NAT
Network Address Translation

3.3.7
POTS
Plain Old Telephone Service

3.3.8
PPP
Point-to-Point Protocol

3.3.9
RSIP
Realm Specific Internet Protocol

3.3.10
RTSP
Real Time Streaming Protocol

3.3.11
SIP
Session Initiation Protocol

3.3.12
SNMP
Simple Network Management Protocol

3.3.13
SOAP
Simple Object Access Protocol

3.3.14
SSDP
Simple Service Discovery Protocol

3.3.15
UI
User Interface

3.3.16
UUID
Universally Unique IDentifier

3.3.17
VC
Virtual Channel

3.3.18
VCI
Virtual Channel Identifier

3.3.19
VPI
Virtual Path Identifier

3.3.20
VPN
Virtual Private Network

4 Notations and conventions

4.1 Notation

- UPnP interface names defined in [1] are styled in **green bold underlined** text.
- UPnP interface names defined outside of [1] are styled in *red italic underlined* text.
- Simple Service Discovery Protocol (SSDP) names are styled in blue underlined text.
- Some additional non-interface names and terms are styled in *italic* text.
- Words that are emphasized are also styled in *italic* text. The difference between italic terms and italics for emphasis will be apparent by context.
- Strings that are to be taken literally are enclosed in “double quotes”.

4.2 Data types

4.2.1 Primary data types

Primary data type definitions come from two sources:

- State variable and action argument data types are defined in [1].
- Basic data types for XML element and attribute values are defined in [12]. The XML data types are essential, since all communications between UPnP services and control points are encoded in XML.

For the UPnP-defined data type **boolean**, it is strongly recommended to use the value “**0**” for false, and the value “**1**” for true. The values “**true**”, “**false**”, “**yes**” and “**no**” are deprecated—they shall not appear in output, but shall be accepted on input.

For the data type `xsd:boolean`, it is strongly recommended to use the value “`0`” for false, and the value “`1`” for true. The values “`true`” and “`false`” are allowed, but not recommended.

4.2.2 Secondary data types

4.2.2.1 CSV lists

The Comma Separated Value list allows a single string to contain multiple values represented in a list form. Lists may either be homogeneous (all values are the same type) or heterogeneous (values of different types are allowed). Lists may also consist of repeated occurrences of homogeneous or heterogeneous subsequences, all of which have the same syntax and semantics (same number of values, same value types and in the same order). The basic data type of a homogeneous list is **string** or `xsd:string` and is denoted by CSV (x), where x is the type of the individual values. The data type of a heterogeneous list is also **string** or `xsd:string` and is denoted by CSV (x, y, z), where x, y and z are the types of the individual values. The data type of a repeated subsequence list is **string** or `xsd:string` and denoted by CSV ({x, y, z}), where x, y and z are the types of the individual values in the subsequence and the subsequence may be repeated zero or more times. If the number of values in a heterogeneous list is too large to show each type individually or to represent as a repeated subsequence, it is denoted by CSV (heterogeneous), and the variable description includes additional information as to the expected sequence of values appearing in the list and their corresponding types.

- The containing string can be either of type **string** or `xsd:string`.
- Commas separate values within a list.
- The escape character is the backslash, “\”, U+005C
  - To represent a comma character within a single value, escape it as “\,”
  - To represent the backslash character as itself, escape it as "\\"
- Integer values are represented in CSVs with the same syntax as the integer data type specified in [1]
- Boolean values are represented in CSVs as “0” for false or “1” for true.
- White space before, after, or interior to any numeric data type is not allowed.
- White space before, after, or interior to any other data type is part of the value.

Table 1 — CSV examples

Type of Refinement String	Value	Comments
CSV (**int**) or CSV (xsd:integer)	“1,-5,006,0,+7”	List of 5 integers.
CSV (**boolean**) or CSV (xsd:Boolean)	“0,1,1,0”	List of 4 booleans
CSV (**string**) or CSV (xsd:string)	“Smith\, Fred,Jones\, Davey”	List of 2 names, “Smith, Fred” and “Jones, Davey”
CSV (**i4**,**string**,**ui2**) or CSV (xsd:int, xsd:string, xsd:unsignedShort)	“-29837, string with leading blanks,0”	Note that the second value is “ string with leading blanks”
CSV (**i4**) or CSV (xsd:int)	“3, 4”	Illegal CSV. White space is not allowed as part of an integer value.
CSV (**string**) or CSV (xsd:string)	“,,”	List of 3 empty string values
CSV ({**string**,**string**,**ui2**}) or CSV ({xsd: string, xsd:string, xsd:unsignedShort})	“Alice,Marketing,5,Sue,R&D,21,Dave,Finance,7”.	List of unspecified number of people and associated attributes. Each person is described by 3 elements: a name, a department and years-of-service

4.2.2.2 Other

There are no other secondary data types than CSV lists.

4.3 Vendor-defined extensions

When vendors add vendor-defined state variables, actions or properties, their assigned names and XML representation shall follow the naming conventions and XML rules for non-standard vendor extensions in [1].

5 Service model

5.1 Service type

A service that complies with this specification shall identify itself with the URN:

urn:schemas-upnp-org:service:*WANIPConnection:2*

All uses of the name *WANIPConnection* in this document specifically refer to version *WANIPConnection:2* of this service type.

5.2 Changes from *WANIPConnection:1*

5.2.1 Backward compatibility

WANIPConnection:2 is backward-compatible with *WANIPConnection:1* ([6]), except where access control has been added.

5.2.2 Generic requirements and other changes

- The NAT behaviour should follow IETF RFCs [15] and [16],
- When a control point creates a port forwarding rule for inbound traffic, this rule shall also be applied when NAT port triggering occurs for outbound traffic,
- UPnP IGD implementing *WANIPConnection:2* shall expose UPnP services only over the LAN interface. The *WANIPConnection:2* implementation shall reject UPnP requests from the WAN interfaces,
- Upon startup, The UPnP IGD Device implementation, implementing the *WANIPConnection:2* service, shall broadcast an ssdp:byebye before sending the initial ssdp:alive onto the local network. Sending an ssdp:byebye as part of the normal start up process for a UPnP device ensures that UPnP control points with information about the previous device instance will safely discard state information about the previous device instance before communicating with the new device instance.
- Deprecated error codes. *WANIPConnection* shall not report the following error codes, though control points are still required to support them for legacy services.
  - 724 *SamePortValuesRequired*
  - 725 *OnlyPermanentLeasesSupported*
  - 726 *RemoteHostOnlySupportsWildcard*
  - 727 *ExternalPortOnlySupportsWildcard*

  For details, see 5.6.16.
- Clarifications to *WANIPConnection:1* ([6]).

5.2.3 New state variables

- *SystemUpdateID* is used to track changes which could affect NAT port mappings,
- *A_ARG_TYPE_Manage* is a parameter used in new actions which allows a control point to request access level elevation,
- *A_ARG_TYPE_PortListing* is a data structure used to return a list of port mappings.

5.2.4 New actions

- *DeletePortMappingRange()* allows removal of a range of port mappings,
- *GetListOfPortMappings()* allows retrieval of a list of existing port mappings,
- *AddAnyPortMapping()* allows the control point to request a specific external port, and if the port is not free the gateway assigns a free port.

5.2.5 Changes in existing actions and procedures

- *PortMappingLeaseDuration* can be either a value between 1 and 604800 seconds or the zero value (for infinite lease time). Note that an infinite lease time can be only set by out-of-band mechanisms like WWW-administration, remote management or local management,
- If a control point uses the value 0 to indicate an infinite lease time mapping, it is required that gateway uses the maximum value instead (e.g. 604800 seconds),
- *WANIPConnection:2* introduces access control features. [3] recommends access control requirements and authorization levels to be applied by *default*. However, services may choose a different security policy,
- *WANIPConnection:2* of the service uses access restriction for certain actions and parameters. In the 2-box model, where the control point is in the same device that desires to receive communication through the NAT, [3] recommends that access control is not needed. But in the 3-box model, where the control point is configuring NAT port mappings for a third device, [3] recommends that authentication and authorization is used.
- To summarize, [3] recommends that unauthenticated and unauthorized control points are only allowed to control port mappings which have *InternalClient* value equals to the control point's IP address. However, services may choose a different security policy,
- [3] recommends that unauthenticated and unauthorized control points are only allowed to invoke actions with *NewExternalPort* and *NewInternalPort* values greater than or equal to 1024. However, services may choose a different security policy,
- NAT port mapping rules can be created by other mechanisms besides *WANIPConnection*. Therefore, it is possible that port mappings done by independent mechanisms may overlap or conflict. It is left to vendors to determine a suitable algorithm to resolve conflicting mappings. A new error code has been created (729 *ConflictWithOtherMechanisms*) in order to allow *WANIPConnection* to deny a request due to conflict with other mechanisms,
- *WANIPConnection* shall support both wildcard and specific IP address values for *RemoteHost* (only the wildcard value was required in *WANIPConnection:1*),
- *WANIPConnection* shall support specific port values for *ExternalPort*,
- The error code 731 *ReadOnly* (indicating that it is not possible to modify the value because it is read only) has been created for the *SetConnectionType()* action,
- The error code 728 *NoPortMapsAvailable* (There are not enough free ports available to complete the port mapping) has been created for the *AddPortMapping()* and *AddAnyPortMapping()* actions,
- The error code 730 *PortMappingNotFound* (There are not port mappings in the specified range) has been created for the *DeletePortMappingRange()* and *GetListOfPortMappings()* actions,
- The error code 732 *WildCardNotPermittedInIntPort* (The internal port cannot be wild-carded) has been created for the *AddPortMapping()* and *AddAnyPortMapping()* actions,
- The error code 733 *InconsistentParameters* (*NewStartPort* and *NewEndPort* values are not consistent) has been created for the *DeletePortMappingRange()* and *GetListOfPortMappings()* actions.

5.3 Service Architecture

5.3.1 Introduction

All IP Internet connections are set up from a WAN interface of the *InternetGatewayDevice* or bridged through the gateway to Internet Service Providers (ISPs). *WANDevice* is a container for all UPnP services associated with a physical WAN device. It is assumed that clients are connected to *InternetGatewayDevice* via a LAN (IP-based network).

An instance of a *WANIPConnection* service is activated (refer to the state variable table) for each actual Internet connection instance on a *WANConnectionDevice*. *WANIPConnection* service provides IP-level connectivity with an ISP for networked clients on the LAN.

In accordance with [1], the maximum number of *WANIPConnection* service instances is static and specified in the *InternetGatewayDevice* description document.

A *WANConnectionDevice* may include a *WAN{POTS/DSL/Cable/Ethernet}LinkConfig* service that encapsulates Internet access properties pertaining to the physical link of a particular WAN access type. These properties are common to all instances of *WANIPConnection* in a *WANConnectionDevice*.

A *WANDevice* provides a *WANCommonInterfaceConfig* service that encapsulates Internet access properties common across all *WANConnectionDevice* instances.

5.3.2 Main feature sets

This service has three main feature sets.

a) The first feature set focuses on managing connections when the connection is not always on. This service is impacted only if the instance of *WANIPConnection* is in *IP_Routed* mode. An IP connection can be started with *RequestConnection()* action, and *RequestTermination()* or *ForceTermination()* can be used to disconnect an active connection. In addition to these actions, it is possible to set the duration for the connection or to set the idle time after which the connection may be terminated automatically.

b) The second feature set includes a number of actions that allow retrieval of status information from the gateway including: getting connection type, disconnect times, external IP address, NAT and RSIP status. These can be used to determine the state of the gateway and its settings.

   The NAT Traversal or port mapping functionality allows creation of mappings for both TCP and UDP protocols between an external IGD port (called *ExternalPort*) and an internal client address associated with one of its ports (respectively called *InternalClient* and *InternalPort*). It is also possible to narrow the mapping by limiting the mapping to a specific remote host[1]. This feature is used to allow external hosts to reach hosts located behind NATs.

   Getting information about existing port mapping can be done with the following actions:

   - *GetGenericPortMappingEntry()* is used to retrieve port mapping entries based on a table index. The table consists of all port mapping entries. The size of the table and its indexes are updated as the number of port mapping entries changes,
   - *GetSpecificPortMappingEntry()* allows retrieval of a port mapping entry based on the *ExternalPort*, the *PortMappingProtocol*, and the *RemoteHost*,
   - *GetListOfPortMappings()* returns a list of port mapping entries based on a range of *ExternalPort* and the *PortMappingProtocol* values.

   Creation and deletion of port mapping entries is done with the following actions:

   - *AddPortMapping()* allows creation of a single port mapping by specifying all parameters in one call, this action returns an error if the mapping is already reserved. Please note that as of *WANIPConnection:2*, it is recommended to use *AddAnyPortMapping()* instead of *AddPortMapping()*,
   - *AddAnyPortMapping()* allows requests for a preferred port mapping; if it is not free, the gateway will make an alternative reservation,
   - *DeletePortMapping()* allows deletion of a single port mapping entry,
   - *DeletePortMappingRange()* allows removal of a range of port mapping entries.

   NAT port mapping rules can be created by other mechanisms besides *WANIPConnection*. Therefore, it is possible that port mappings done by independent mechanisms may overlap or conflict. It is left to vendors to determine suitable algorithm on how to resolve conflicting

[1] When the *RemoteHost* parameter corresponds to a wildcard value, the port mapping rule has an endpoint independent filtering behaviour, and when the *RemoteHost* parameter corresponds to a specific remote host, the port mapping rule has an address dependent filtering behaviour (see [13] terminology).

mappings. A new error code has been created (729 *PortMappingNotAllowed*) in order to allow *WANIPConnection* to deny a request due to conflict with other mechanisms.

c) The third feature set covers security policies

The *WANIPConnection* service defines security policy classifying actions at two levels:

- Actions that require specific authentication. The access control requirements are specifically referenced within each action's description,
- Access restrictions to actions and parameters based on the control point's IP address. The restrictions are handled locally within this service. In port mappings, there are three UPnP entities involved: The *IGD control point* which edits port mappings, the *client* which needs the port mapping, and the *gateway* service that executes the port mapping. When the control point and the client are located in the same physical device, this is called the 2-box model. If the control point is located in another device, then this is called the 3-box model. [3] recommends that no authentication or authorization is needed in 2-box model in most actions. [3] recommends that control point configuring the gateway is authenticated and authorized in 3-box model. However, services may choose a different security policy.

For security reasons, UPnP IGD shall expose UPnP services only over the LAN interface. IGD shall reject UPnP requests from the WAN interfaces.

5.4 State variables

5.4.1 State variable overview

There are three kinds of state variables in *WANIPConnection:2*:

- Basic connection settings and connections status,
- Disconnection time settings,
- Specific variables for NAT traversal / port mapping creation and deletion.

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

Table 2 — State Variables

Variable Name	R/A	Data Type	Reference
ConnectionType	*R*	**string**	See 5.4.2
PossibleConnectionTypes	*R*	**string** (CSV)	See 5.4.3
ConnectionStatus	*R*	**string**	See 5.4.4
Uptime	*R*	**ui4**	See 5.4.5
LastConnectionError	*R*	**string**	See 5.4.6
AutoDisconnectTime	*A*	**ui4**	See 5.4.7
IdleDisconnectTime	*A*	**ui4**	See 5.4.8
WarnDisconnectDelay	*A*	**ui4**	See 5.4.9
RSIPAvailable	*R*	**boolean**	See 5.4.10
NATEnabled	*R*	**boolean**	See 5.4.11
ExternalIPAddress	*R*	**string**	See 5.4.12
PortMappingNumberOfEntries	*R*	**ui2**	See 5.4.13
PortMappingEnabled	*R*	**boolean**	See 5.4.14
PortMappingLeaseDuration	*R*	**ui4**	See 5.4.15

Variable Name	R/A	Data Type	Reference
RemoteHost	*R*	**string**	See 5.4.16
ExternalPort	*R*	**ui2**	See 5.4.17
InternalPort	*R*	**ui2**	See 5.4.18
PortMappingProtocol	*R*	**string**	See 5.4.19
InternalClient	*R*	**string**	See 5.4.20
PortMappingDescription	*R*	**string**	See 5.4.21
SystemUpdateID	*R*	**ui4**	See 5.4.22
A_ARG_TYPE_Manage	*R*	**boolean**	See 5.4.23
A_ARG_TYPE_PortListing	*R*	**string** (XML fragment)	See 5.4.24
Non standard state variables implemented by an UPnP vendor go here	*X*	*TBD*	*TBD*

Figure 1 shows the relationships among gateway system components and *WANIPConnection* state variables for NAT processing. The relationships are the same in both the 2-box model and 3-box model.

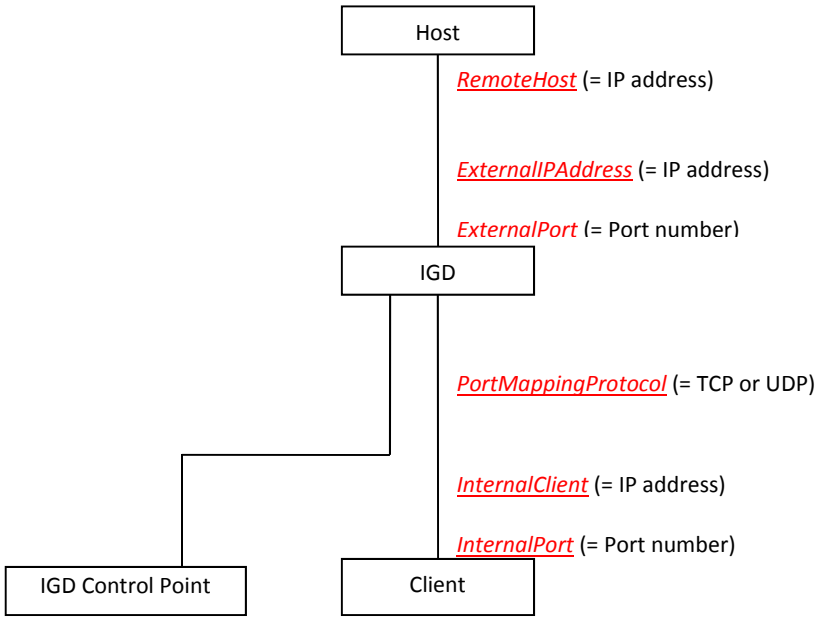

Figure 1 — UPnP IGD component relationships for NAT processing

NAT-related state variables:

- *RemoteHost* is the WAN IP address (destination) of connections initiated by a client in the local network.
- *ExternalIPAddress* is the WAN IP address of the Client as seen by the remote host.
- *ExternalPort* is a source TCP or UDP port number of the Client as seen by the remote host.
- *InternalClient* is the local IP address of the client.
- *InternalPort* is the local TCP or UDP port number of the client.

- *PortMappingProtocol* is either TCP or UDP.

5.4.2 *ConnectionType*

This state variable is a **string** that contains information on the connection types used in the gateway.

Table 3 — allowedValueList for the *ConnectionType* state variable

Value	R/A
Unconfigured	*R*
IP_Routed (DEFAULT)	*R*
IP_Bridged	*R*
Vendor-defined	*X*

5.4.2.1 *Unconfigured*

Valid connection types cannot be identified. This may be due to the fact that the *LinkType* variable (if specified in the *WAN*LinkConfig* service) is uninitialized.

THIS VALUE IS DEPENDENT ON THE DEPLOYMENT AND TESTING SHOULD BE DEFERRED TO THE VENDOR.

5.4.2.2 *IP_Routed*

The Internet Gateway is an IP router between the LAN and the WAN connection.

THIS VALUE IS ONLY APPLICABLE FOR AN IGD DEVICE SUPPORTING NAT. IT SHOULD NOT BE TESTED IN OTHER DEVICE CONFIGURATIONS.

5.4.2.3 *IP_Bridged*

The Internet Gateway is an Ethernet bridge between the LAN and the WAN connection. A router at the other end of the WAN connection from the IGD routes IP packets.

THIS VALUE IS ONLY APPLICABLE FOR AN IGD DEVICE CONFIGURED AS AN ETHERNET BRIDGE. IT SHOULD NOT BE TESTED IN OTHER DEVICE CONFIGURATIONS.

5.4.3 *PossibleConnectionTypes*

This variable represents a CSV list indicating the types of connections possible in the context of a specific modem and link type. Possible values are a subset or proper subset of values listed in Table 3.

Refer to the *WANConnectionDevice* specification for valid combinations of *LinkType* and *PossibleConnectionTypes* for different modems that can support IP based connections.

5.4.4 *ConnectionStatus*

This state variable is a **string** that contains information on the status of the connection.

Table 4 — allowedValueList for the *ConnectionStatus* state variable

Value	R/A
Unconfigured	*R*
Connecting	*A*
Connected	*R*
PendingDisconnect	*A*
Disconnecting	*A*
Disconnected	*R*
Vendor-defined	*X*

5.4.4.1 *Unconfigured*

This value indicates that other variables in the service table are uninitialized or in an invalid state. Examples of such variables include *PossibleConnectionTypes* and *ConnectionType*.

5.4.4.2 *Connecting*

The *WANConnectionDevice* is in the process of initiating a connection for the first time after the connection became disconnected.

5.4.4.3 *Connected*

At least one client has successfully initiated an Internet connection using this instance.

5.4.4.4 *PendingDisconnect*

The connection is active (packets are allowed to flow through), but will transition to *Disconnecting* state after a certain period (indicated by *WarnDisconnectDelay*).

5.4.4.5 *Disconnecting*

The *WANConnectionDevice* is in the process of terminating a connection. On successful termination, *ConnectionStatus* transitions to *Disconnected*.

5.4.4.6 *Disconnected*

No ISP connection is active (or being activated) from this connection instance. No packets are transiting the gateway.

5.4.5 *Uptime*

The variable *Uptime* represents time in seconds that this connections has stayed up. The type of this variable is **ui4**.

5.4.6 *LastConnectionError*

This variable is a **string** that provides information about the cause of failure for the last connection setup attempt. The restricted list of enumeration values are listed in Table 5.

Table 5 — allowedValueList for the *LastConnectionError* state variable

Value	R/A
ERROR_NONE	*R*
ERROR_COMMAND_ABORTED	*A*
ERROR_NOT_ENABLED_FOR_INTERNET	*A*
ERROR_ISP_DISCONNECT.	*A*
ERROR_USER_DISCONNECT	*A*
ERROR_IDLE_DISCONNECT	*A*
ERROR_FORCED_DISCONNECT	*A*
ERROR_NO_CARRIER	*A*
ERROR_IP_CONFIGURATION	*A*
ERROR_UNKNOWN	*A*
Vendor-defined	*X*

5.4.7 *AutoDisconnectTime*

The *AutoDisconnectTime* variable represents time in seconds (since the establishment of the connection – measured from the time *ConnectionStatus* transitions to *Connected*), after which connection termination is automatically initiated by the gateway. This occurs irrespective of whether the connection is being used or not. A value of zero for *AutoDisconnectTime* indicates that the connection is not to be turned off automatically. However, this may be overridden by:

- An implementation specific WAN/Gateway service policy,

- *EnabledForInternet* variable (see *WANCommonInterfaceConfig**) being set to "**0**" (false) by a control point,
- Connection termination initiated by ISP.

If *WarnDisconnectDelay* is non-zero, the connection state is changed to *PendingDisconnect*. It stays in this state for *WarnDisconnectDelay* seconds (if no connection requests are made) before switching to *Disconnected*. The data type of this variable is **ui4**.

5.4.8 *IdleDisconnectTime*

IdleDisconnectTime represents the idle time of a connection in seconds (since the establishment of the connection), after which connection termination is initiated by the gateway. A value of zero for this variable allows infinite idle time – connection will not be terminated due to idle time.

Notice that: Layer 2 heartbeat packets are included as part of an idle state i.e., they do not reset the idle timer. The data type of this variable is **ui4**.

If *WarnDisconnectDelay* is non-zero, the connection state is changed to *PendingDisconnect*. It stays in this state for *WarnDisconnectDelay* seconds (if no connection requests are made) before switching to *Disconnected*.

5.4.9 *WarnDisconnectDelay*

This variable represents time in seconds the *ConnectionStatus* remains in the *PendingDisconnect* state before transitioning to *Disconnecting* state to drop the connection. For example, if this variable was set to 5 seconds, and one of the clients terminates an active connection, the gateway will wait (with *ConnectionStatus* as *PendingDisconnect*) for 5 seconds before actual termination of the connection. A value of zero for this variable indicates that no warning will be given to clients before terminating the connection. The data type of this variable is **ui4**.

5.4.10 *RSIPAvailable*

This variable indicates if Realm-specific IP (RSIP) is available as a feature on the Internet Gateway Device. The type of this variable is **boolean**. RSIP has been defined by IETF ([13], [14]) to allow host-NATing using a standard set of message exchanges. It also allows end-to-end applications that otherwise break if NAT is introduced (e.g. IPsec-based VPNs). A gateway that does not support RSIP MUST set this variable to 0.

5.4.11 *NATEnabled*

This **boolean** type variable indicates if Network Address Translation (NAT) is enabled for this connection.

5.4.12 *ExternalIPAddress*

ExternalIPAddress is a **string** containing the external IP address used by NAT for the connection. The format of this string is standard IP address representation and shall be formatted as:

- a set of four decimal digit groups separated by "." as defined in [9],
- or an empty string.

When the external IP address could not be retrieved by the gateway (for example, because the interface is down or because there was a failure in the last connection setup attempt), then the *ExternalIPAddress* shall be equal to the empty string.

5.4.13 *PortMappingNumberOfEntries*

This variable indicates the number of NAT port mapping entries (number of elements in the array) configured on this connection.

5.4.14 *PortMappingEnabled*

This variable allows security conscious users to disable and enable dynamic NAT port mappings on the IGD. The type of this variable is **boolean**.

5.4.15 *PortMappingLeaseDuration*

Table 6 — allowedValueRange for the *PortMappingLeaseDuration* state variable

	Value (in seconds)	**R/A**
minimum	*0*	*R*
maximum	*604800*	*R*
default	*Vendor-defined* (recommended value is 3600)	*R*

This variable determines the lifetime in seconds of a port-mapping lease. Non-zero values indicate the duration after which a port mapping will be removed, unless a control point refreshes the mapping.

In *WANIPConnection:1*, a value of 0 was used to create a static port mapping. In *WANIPConnection:2*, it is no longer possible to create static port mappings via UPnP actions. Instead, an out-of-band mechanism is required to do so (see WWW-administration, remote management or local management). In order to be backward compatible with legacy control points, the value of 0 shall be interpreted as the maximum value (e.g. 604800 seconds, which corresponds to one week).

Notice that: Port mappings are not required to be persistent across device resets or reboots. It is up to the control points to recreate their port mappings when needed.

5.4.16 *RemoteHost*

This variable represents the source of inbound IP packets. This variable can contain a host name or a standard IPv4 address representation. This state variable shall be formatted as:

- a domain name of a network host like it is defined in [8],
- or as a set of four decimal digit groups separated by "." as defined in [9],
- or an empty string.

This will be a wildcard in most cases (an empty string). As of *WANIPConnection:2*, NAT vendors are required to support non-wildcarded IP addresses in addition to wildcards. A non-wildcard value will allow for “narrow” port mappings, which may be desirable in some usage scenarios. When *RemoteHost* is a wildcard, all traffic sent to the *ExternalPort* on the WAN interface of the gateway is forwarded to the *InternalClient* on the *InternalPort* (this corresponds to the endpoint independent filtering behaviour defined in the [16]). When *RemoteHost* is specified as a specific external IP address as opposed to a wildcard, the NAT will only forward inbound packets from this *RemoteHost* to the *InternalClient*. All other packets will be dropped (this corresponds to the address dependent filtering behaviour defined in [16]).

5.4.17 *ExternalPort*

This variable is of type **ui2** and represents the external port that the NAT gateway would “listen” on for connection requests to a corresponding *InternalPort* on an *InternalClient*. Inbound packets to this external port on the WAN interface of the gateway should be forwarded to *InternalClient* on the *InternalPort* on which the message was received. If this value is specified as a wildcard (i.e. 0), connection request on all external ports (that are not otherwise mapped) will be forwarded to *InternalClient*. In the wildcard case, the value(s) of *InternalPort* on *InternalClient* are ignored by the IGD for those connections that are forwarded to *InternalClient*. Obviously only one such entry can exist in the NAT at any time and conflicts are handled with a “first write wins” behavior. As of *WANIPConnection:2*, NAT vendors are required to support non-wildcarded ports.

5.4.18 *InternalPort*

This variable is of type **ui2** and represents the port on *InternalClient* that the gateway should forward connection requests to. A value of 0 is not allowed. NAT implementations that do not permit different values for *ExternalPort* and *InternalPort* will return an error.

Table 7 — allowedValueRange for the *InternalPort* state variable

	Value	R/A
minimum	*1*	*R*
maximum	*65535*	*R*

5.4.19 *PortMappingProtocol*

This **string** variable represents the protocol of the port mapping. Possible values are *TCP* or *UDP*.

Table 8 — allowedValueList for the *PortMappingProtocol* state variable

Value	R/A
TCP	*R*
UDP	*R*
Vendor-defined	*X*

5.4.20 *InternalClient*

This variable is a string containing the IP address or DNS host name of an *InternalClient* (on the residential LAN). This variable can contain a host name or a standard IPv4 address representation. This state variable shall be formatted as:

- a domain name of a network host like it is defined in [8],
- or as a set of four decimal digit groups separated by "." as defined in [9].

Note that if the gateway does not support DHCP, it does not have to support DNS host names. Consequently, support for an IP address is required, and support for DNS host names is recommended. This value cannot be a wildcard (i.e. empty string). It shall be possible to set the *InternalClient* to the broadcast IP address 255.255.255.255 for UDP mappings. This is to enable multiple NAT clients to use the same well-known port simultaneously.

5.4.21 *PortMappingDescription*

This is a **string** representation of a port mapping. The format of the description string is not specified and is application dependent. If specified, the description string can be displayed to a user via the UI of a control point, enabling easier management of port mappings. The description string for a port mapping (or a set of related port mappings) is not required to be unique across multiple instantiations of an application on multiple nodes in the residential LAN.

5.4.22 *SystemUpdateID*

The type of this variable is **ui4,** and it is used to notify of changes done in NAT or firewall rules.

EXAMPLES:

- the user changed the firewall level settings of his IGD, and the NAT port mappings rules are no more valid,
- the user disabled the UPnP IGD NAT traversal facilities through the WWW-administration of the IGD,
- the user updated a NAT rule thanks to the WWW-administration of his IGD, and that NAT rule was previously created by a UPnP IGD control point.

Whenever a change is done, the value of this variable is incremented by 1 and evented. A change can be an addition, a removal, an update, or the fact that a rule is disabled or enabled. So, control points are encouraged to check if their port mappings are still valid when notified.

This variable is evented when something which affects the port mappings validity occurs. Even if the event affects several port mappings rules, the variable is evented once (and not for each impacted port mappings rules).

Moreover, a control point needs to detect if the IGD has rebooted in order to check if its NAT port mappings rules are still valid. In *UPnP Device Architecture 1.0*, [1] there are no mechanisms to detect when a device has rebooted. In *UPnP Device Architecture 1.1*, [2], it is possible to detect rebooting thanks to **BOOTID**. However, *UPnP Device Architecture 1.1* is not required for implementing the *WANIPConnection* service.

Therefore, in order to allow control points to distinguish between the reboot of the IGD and the refreshment of the advertisement, the UPnP IGD shall broadcast an ssdp:byebye before sending the initial ssdp:alive onto the local network upon startup. Sending an ssdp:byebye as part of the normal start up process for a UPnP device ensures that UPnP control points with information about the previous device instance will safely discard state information about the previous device instance before communicating with the new device instance.

5.4.23 *A_ARG_TYPE_Manage*

This argument type is used to describe management intent when issuing certain actions with elevated level of access. The type of this argument is **boolean**.

5.4.24 *A_ARG_TYPE_PortListing*

This argument type contains the list of port mapping entries.

5.4.24.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument shall validate against the XML schema for *PortMappingList* in the XML namespace "**urn:schemas-upnp-org:***gw:WANIPConnection*" which is located at: "http://www.upnp.org/schemas/gw/WANIPConnection-v2.xsd".

5.4.24.1.1 Description of fields in the *PortMappingList* structure

The fields in the XML fragment are:

PortMappingList is a required structure defined as an XML element. There shall be only one element of this kind.

PortMappingEntry is a required structure defined as an XML element. There can be zero to 65535 elements of this kind.

NewRemoteHost is a required structure defined as an XML attribute. Its type corresponds to the *RemoteHost* state variable (type **string**).

NewExternalPort is a required structure defined as an XML attribute. Its type corresponds to the *ExternalHost* state variable (type **ui2**).

NewProtocol is a required structure defined as XML attribute. Its type corresponds to the *PortMappingProtocol* state variable (type **string**).

NewInternalPort is a required structure defined as XML attribute. Its type corresponds to the *InternalPort* state variable (type **ui2**).

NewInternalClient is a required structure defined as XML attribute. Its type corresponds to the *InternalClient* state variable (type **string**).

NewEnabled is a required structure defined as XML attribute. Its type corresponds to the *PortMappingEnabled* state variable (type **boolean**).

NewDescription is a required structure defined as XML attribute. Its type corresponds to the *PortMappingDescription* state variable (type **string**).

NewLeaseTime is a required structure defined as XML attribute. Its type corresponds to the *PortMappingLeaseDuration* state variable (type **ui4**).

5.4.24.2 Sample XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<p:PortMappingList xmlns:p="urn:schemas-upnp-org:gw:WANIPConnection"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schemas-upnp-org:gw:WANIPConnection
http://www.upnp.org/schemas/gw/WANIPConnection-v2.xsd">
  <p:PortMappingEntry>
    <p:NewRemoteHost>202.233.2.1</p:NewRemoteHost>
    <p:NewExternalPort>2345</p:NewExternalPort>
    <p:NewProtocol>TCP</p:NewProtocol>
    <p:NewInternalPort>2345</p:NewInternalPort>
    <p:NewInternalClient>192.168.1.137</p:NewInternalClient>
    <p:NewEnabled>1</p:NewEnabled>
    <p:NewDescription>dooom</p:NewDescription>
    <p:NewLeaseTime>345</p:NewLeaseTime>
  </p:PortMappingEntry>
  <p:PortMappingEntry>
    <p:NewRemoteHost>134.231.2.11</p:NewRemoteHost>
    <p:NewExternalPort>12345</p:NewExternalPort>
    <p:NewProtocol>TCP</p:NewProtocol>
    <p:NewInternalPort>12345</p:NewInternalPort>
    <p:NewInternalClient>192.168.1.137</p:NewInternalClient>
    <p:NewEnabled>1</p:NewEnabled>
    <p:NewDescription>dooom</p:NewDescription>
    <p:NewLeaseTime>345</p:NewLeaseTime>
  </p:PortMappingEntry>
</p:PortMappingList>
```

Note that this XML fragment is returned as part of SOAP response, and so needs to be XML escaped. Moreover, note that the XML declaration, `<?xml version="1.0" encoding="UTF-8"?>`, is allowed.

5.5 Eventing and Moderation

Table 9 — Eventing and Moderation

Variable Name	Evented	Moderated	Criteria
ConnectionType	*NO*	*NO*	N/A
PossibleConnectionTypes	*YES*	*NO*	N/A
ConnectionStatus	*YES*	*NO*	N/A
Uptime	*NO*	*NO*	N/A
LastConnectionError	*NO*	*NO*	N/A
AutoDisconnectTime	*NO*	*NO*	N/A
IdleDisconnectTime	*NO*	*NO*	N/A
WarnDisconnectDelay	*NO*	*NO*	N/A
RSIPAvailable	*NO*	*NO*	N/A
NATEnabled	*NO*	*NO*	N/A
ExternalIPAddress	*YES*	*NO*	N/A
PortMappingNumberOfEntries	*YES*	*NO*	N/A
PortMappingEnabled	*NO*	*NO*	N/A
PortMappingLeaseDuration	*NO*	*NO*	N/A
RemoteHost	*NO*	*NO*	N/A

Variable Name	Evented	Moderated	Criteria
ExternalPort	*NO*	*NO*	N/A
InternalPort	*NO*	*NO*	N/A
PortMappingProtocol	*NO*	*NO*	N/A
InternalClient	*NO*	*NO*	N/A
PortMappingDescription	*NO*	*NO*	N/A
SystemUpdateID	*YES*	*NO*	N/A
A_ARG_TYPE_Manage	*NO*	*NO*	N/A
A_ARG_TYPE_PortListing	*NO*	*NO*	N/A
Non-standard state variables implemented by an UPnP vendor go here.	*TBD*	*TBD*	*TBD*

5.5.1 Eventing of *PossibleConnectionTypes*

This event is created whenever the list of *PossibleConnectionTypes* has changed.

5.5.2 Eventing of *ConnectionStatus*

This variable is evented whenever *ConnectionStatus* has changed.

Implementations are not required to event intermediate states of a connection transition.

5.5.3 Eventing of *ExternalIPAddress*

This variable is evented whenever the external IP address of the gateway has changed.

5.5.4 Eventing of *PortMappingNumberOfEntries*

This variable is evented when the number of port mapping entries changes. *SystemUpdateID* and *PortMappingNumberOfEntries* shall be evented at the same time when mapping rules are added or removed.

5.5.5 Eventing of *SystemUpdateID*

This variable is evented when NAT or firewall rules have been changed. The eventing of this variable shall be done in the same time when *PortMappingNumberofEntries* is evented.

5.5.6 Relationships among State Variables

If *ConnectionStatus* is set to *Unconfigured*, all other variables are set to their default values.

If *ConnectionStatus* is set to *Disconnected*, *Uptime* is set to its default value.

If *NATEnabled* is set to 0, other port mapping related set actions are essentially disabled. Get actions may still succeed. Moreover, *SystemUpdateID* is incremented by 1 and evented.

For port mappings, the *PortMappingLeaseDuration* variable counts down from the value set by the *AddPortMapping()* or *AddAnyPortmapping()* action. The value counts down independent of the state of *PortMappingEnabled* for that specific port mapping. If a *GetGenericPortMappingEntry()* or *GetSpecificPortMappingEntry()* action is invoked, the remaining time on a port-mapping lease is returned to the control point. For example if a port mapping is added with a lease duration of 1500 seconds and *GetSpecificPortMappingEntry()* is invoked on that port mapping 500 seconds later, *PortMappingLeaseDuration* will return 1000 as its value (+/- a few seconds accounting for clock drift). When *PortMappingLeaseDuration* counts to zero, the entry will be deleted by the *WANIPConnection*, independent of the state of *PortMappingEnabled* for that specific port mapping. The *WANIPConnection* will correspondingly modify local NAT (and firewall settings if appropriate) to stop forwarding packets as was specified in the deleted port mapping. This will also cause *PortMappingNumberOfEntries* to be decremented by 1 and evented. *SystemUpdateID* is incremented by 1 and evented every time a change is made to port mappings. Port mappings will not be automatically reinitiated by the *WANIPConnection*—it is the responsibility of a control point to refresh the port mapping a few “threshold” seconds before the port mapping is set to expire (i.e. *PortMappingLeaseDuration* equals zero) to prevent service disruption. The value of “threshold” seconds is implementation dependent.

5.6 Actions

Table 10 — Actions

Name	Device R/A [a]	Control Point R/A [b]
SetConnectionType()	R	A
GetConnectionTypeInfo()	R	A
RequestConnection()	R	A
RequestTermination()	A	A
ForceTermination()	R	A
SetAutoDisconnectTime()	A	A
SetIdleDisconnectTime()	A	A
SetWarnDisconnectDelay()	A	A
GetStatusInfo()	R	R
GetAutoDisconnectTime()	A	A
GetIdleDisconnectTime()	A	A
GetWarnDisconnectDelay()	A	A
GetNATRSIPStatus()	R	R
GetGenericPortMappingEntry()	R	R
GetSpecificPortMappingEntry()	R	R
AddPortMapping()	R	R
AddAnyPortMapping()	R	R
DeletePortMapping()	R	R
DeletePortMappingRange()	R	A
GetExternalIPAddress()	R	R
GetListOfPortMappings()	*R*	*A*
Non-standard actions implemented by an UPnP vendor go here.	*X*	*X*

a This column specifies the requirements for implementing the action in a service.

b This column specifies the requirements for supporting the action in a control point.

Table 11 — Common parameters

Variable Name	Related state variable	Description
NewRemoteHost	*RemoteHost*	This argument refers to the remote host which sends packets to the IGD. If a wildcard is defined then packets may be sent by any host, but if a specific IP address is defined, then only packets sent by this IP address are forwarded. This argument is of type **string** in format of “x.x.x.x”. The wildcard value corresponds to an empty string. This may also be DNS name in dotted notation. See *RemoteHost* state variable definition for details.
NewExternalPort	*ExternalPort*	This argument defines the external port value of a port mapping. This port number is visible to external hosts during NAT operation. This argument is of type **ui2**.
NewProtocol	*PortMappingProtocol*	This argument defines whether this port mapping is used by TCP or UDP connection. This argument is type of **string** and allowed values are “*TCP*” and “*UDP*”.

Variable Name	Related state variable	Description
NewInternalPort	*InternalPort*	This argument defines the port number of the internal client where IGD forwards the incoming traffic. This argument is of type **ui2**.
NewInternalClient	*InternalClient*	This argument defines the IP address of the internal host where IGD forwards incoming traffic. This argument is of type **string** in format of "x.x.x.x". This may also be DNS name in dotted notation. See *InternalClient* state variable definition for details.
NewEnabled	*PortMappingEnabled*	This argument defines if this specific port mapping is enabled or disabled. This argument is type of **boolean**.
NewPortMappingDescription	*PortMappingDescription*	This argument provides a description of a port mapping entry. The type is **string** and formatting is left to applications.
NewLeaseDuration	*PortMappingLeaseDuration*	This argument defines the duration of the port mapping. The value of this argument shall be greater than 0. A *NewLeaseDuration* with value 0 means static port mapping, but static port mappings can only be created through an out-of-band mechanism. If this parameter is set to 0, default value of 604800 MUST be used. The recommended value for *NewLeaseDuration* is 3600 seconds.
NewStartPort	*ExternalPort*	This argument is type of **ui2,** and it is used to describe the beginning of a port mapping range. Its allowed value range is the same as for *ExternalPort* state variable, but it shall be less than or equal to *NewEndPort*.
NewEndPort	*ExternalPort*	This argument is type of **ui2,** and it is used to describe the end of a port mapping range. Its allowed value range is the same as for *ExternalPort* state variable, but it shall be greater than or equal to *NewStartPort*.
NewManage	*A_ARG_TYPE_Manage*	This argument is type of **boolean**, and if a control point wants to get or to remove only its own port mappings it should be set to "**0**" (false). If the intent is to manage port mappings for other clients, then *NewManage* should be set to "**1**" (true). This flag does not supersede access control based on control points IP address.

5.6.1 *SetConnectionType()*

This action sets up a specific connection type. Clients on the LAN may initiate or share a connection only after this action completes and *ConnectionType* is set to a value other than *Unconfigured*. *ConnectionType* can be a read-only variable in cases where some form of auto configuration is employed.

5.6.1.1 Arguments

Table 12 — Arguments for *SetConnectionType()*

Argument	Direction	relatedStateVariable
NewConnectionType	*IN*	*ConnectionType*

5.6.1.2 *NewConnectionType*

This argument defines what type of connection is to be established. See definition in subclause 5.4.2.

5.6.1.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

If the control point uses a *NewConnectionType* value which is not in the allowedValueList of the *ConnectionType* state variable, the gateway shall return the 601 *Argument Value Out of Range* error code (defined in [1]).

5.6.1.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.1.5 Dependency on Device State

If *ConnectionStatus* is neither *Disconnected* nor *Unconfigured*, this action cannot be completed.

5.6.1.6 Effect on Device State

This action sets the connection to a specific type. No connections can be established if *ConnectionType* is set to *Unconfigured*.

5.6.1.7 Errors

Table 13 — Error Codes for *SetConnectionType()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in .
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.
703	*InactiveConnectionStateRequired*	Current value of *ConnectionStatus* should be either *Disconnected* or *Unconfigured* to permit this action.
731	*ReadOnly*	Not possible to modify the value because it is read only.

5.6.2 *GetConnectionTypeInfo()*

This action retrieves the values of the current connection type and allowable connection types.

5.6.2.1 Arguments

Table 14 — Arguments for *GetConnectionTypeInfo()*

Argument	Direction	relatedStateVariable
NewConnectionType	*OUT*	*ConnectionType*
NewPossibleConnectionTypes	*OUT*	*PossibleConnectionTypes*

5.6.2.2 *NewConnectionType*

This argument is defined in subclause 5.4.2.

5.6.2.3 *NewPossibleConnectionTypes*

This argument is defined in subclause 5.4.3.

5.6.2.4 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.2.5 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.2.6 Dependency on Device State

None.

5.6.2.7 Effect on Device State

None.

5.6.2.8 Errors

Table 15 — Error Codes for *GetConnectionTypeInfo()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.3 *RequestConnection()*

A client sends this action to initiate a connection on an instance of a connection service that has a configuration already defined. *RequestConnection()* causes the *ConnectionStatus* to immediately change to *Connecting* (if implemented) unless the action is not permitted in the current state of the IGD or the specific service instance. This change of state will be evented. *RequestConnection()* should synchronously return at this time in accordance with UPnP architecture requirements that mandate that an action can take no more than 30 seconds to respond synchronously. However, the actual connection setup may take several seconds more to complete. If the connection setup is successful, *ConnectionStatus* will change to *Connected* and will be evented. If the connection setup is not successful, *ConnectionStatus* will eventually revert back to *Disconnected* and will be evented. *LastConnectionError* will be set appropriately in either case. While this might be obvious, it is worth noting that a control point shall not source packets to the Internet until *ConnectionStatus* is updated to *Connected*, or the IGD may drop packets until it transitions to the *Connected* state.

The process of requesting a connection is described in subclause 5.7.1.

5.6.3.1 Arguments

None.

5.6.3.2 Service Requirements

The IGD should implement a timeout mechanism to ensure that it does not remain in the *Connecting* state forever. The timeout value is implementation dependent.

When *ConnectionStatus* is in *PendingDisconnect* state, if any client sends *RequestConnection()* command, the gateway may choose to discontinue the termination process by changing *ConnectionStatus* to *Connected*. If connection is not restored, the gateway will return error code indicating that the connection was in the process of being torn down.

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.3.3 Control Point Requirements When Calling The Action

Control points should manage a timeout for initiated connections to recover from catastrophic failures on the IGD. The timeout value is implementation dependent.

The IGD may take several seconds (or even a few minutes) to transition from the *Connecting* state to the *Connected* state. Control points should moderate the polling frequency of the *ConnectionStatus* variable on the IGD so as to not create data storms on the network.

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.3.4 Dependency on Device State

The gateway shall be configured before attempting connection. The state variable *ConnectionStatus* shall be equal to either *Disconnected*, *PendingDisconnect* or *Connected* value, and the state variable *ConnectionType* shall be equal to *IP_Routed*.

The *EnabledForInternet* flag in *WANCommonInterfaceConfig* shall be set to "**1**" (true).

5.6.3.5 Effect on Device State

If successful, *ConnectionStatus* is changed to *Connected*.

5.6.3.6 Errors

Table 16 — Error Codes for *RequestConnection()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.
704	*ConnectionSetupFailed*	There was a failure in setting up the IP or PPP connection with the service provider.
705	*ConnectionSetupInProgress*	The connection is already in the process of being setup (Current *ConnectionStatus* is *Connecting*).
706	*ConnectionNotConfigured*	Current *ConnectionStatus* is *Unconfigured*.
707	*DisconnectInProgress*	The connection is in the process of being torn down (Current *ConnectionStatus* is *Disconnecting*).
708	*InvalidLayer2Address*	Corresponding Link Config service has an invalid VPI/VCI or phone number.
709	*InternetAccessDisabled*	The *EnabledForInternet* flag is set to "**0**" (false).
710	*InvalidConnectionType*	This action is not permitted for the specified *ConnectionType* (Current *ConnectionType* is *Unconfigured* or *IP_Bridged*).

5.6.4 *RequestTermination()*

A client may send this command to any connection instance in *Connected* or *Connecting* state to change *ConnectionStatus* to *Disconnected* (through the *Disconnecting* state, if implemented). Connection state changes to *PendingDisconnect* (if implemented) depend on the value of *WarnDisconnectDelay* variable. Connection termination will depend on whether other clients intend to continue to use the connection by invoking the *RequestConnection()* action. The process of terminating a connection is described in subclause 5.7.2.

5.6.4.1 Arguments

None.

5.6.4.2 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control

point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.4.3 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.4.4 Dependency on Device State

ConnectionType shall be equal to *IP_Routed* to allow this action to work.

ConnectionStatus shall be equal to *Connected* or *Connecting* to allow this action to work.

If the connection state is *Connecting* when a client issues a *RequestTermination()*, the state transitions to *Disconnected* directly – it does not go to *PendingDisconnect* even if *WarnDisconnectDelay* is nonzero.

5.6.4.5 Effect on Device State

If successful, *ConnectionStatus* is changed to *Disconnected*.

5.6.4.6 Errors

Table 17 — Error Codes for *RequestTermination()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.
706	*ConnectionNotConfigured*	Current *ConnectionStatus* is *Unconfigured*.
707	*DisconnectInProgress*	The connection is in the process of being torn down (Current *ConnectionStatus* is *Disconnecting* or *PendingDisconnect*).
710	*InvalidConnectionType*	This command is valid only when *ConnectionType* is *IP_Routed*.
711	*ConnectionAlreadyTerminated*	An attempt was made to terminate a connection that is no longer active (Current *ConnectionStatus* is *Disconnected*).

5.6.5 *ForceTermination()*

A client may send this command to any connection instance in *Connected*, *Connecting* or *PendingDisconnect* state to change *ConnectionStatus* to *Disconnected* (goes via *Disconnecting* state if implemented). Connection state immediately transitions to *Disconnecting* (or to *Disconnected* if *Disconnecting* is not implemented) irrespective of the setting of *WarnDisconnectDelay* variable. The process of terminating a connection is described in subclause 5.7.2.

5.6.5.1 Arguments

None.

5.6.5.2 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.5.3 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.5.4 Dependency on Device State

ConnectionType shall be equal to *IP_Routed* to allow this action to work.

ConnectionStatus shall be equal to *Connected*, *Connecting* or *PendingDisconnect* to allow this action to work.

5.6.5.5 Effect on Device State

If successful, *ConnectionStatus* is changed to *Disconnected*.

5.6.5.6 Errors

Table 18 — Error Codes for *ForceTermination()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.
706	*ConnectionNotConfigured*	Current *ConnectionStatus* is *Unconfigured*.
707	*DisconnectInProgress*	The connection is in the process of being torn down (Current *ConnectionStatus* is *Disconnecting*).
710	*InvalidConnectionType*	This command is valid only when *ConnectionType* is *IP_Routed*.
711	*ConnectionAlreadyTerminated*	An attempt was made to terminate a connection that is no longer active (Current *ConnectionStatus* is *Disconnected*).

5.6.6 *SetAutoDisconnectTime()*

This action sets the time (in seconds) after which an active connection is automatically disconnected. The actual disconnect will occur after *WarnDisconnectDelay* time elapses (if implemented). The process of terminating a connection is described in subclause 5.7.2.

5.6.6.1 Arguments

Table 19 — Arguments for *SetAutoDisconnectTime()*

Argument	Direction	relatedStateVariable
NewAutoDisconnectTime	*IN*	*AutoDisconnectTime*

5.6.6.2 *NewAutoDisconnectTime*

This argument sets the autodisconnect time for the connection. See definition in subclause 5.4.7.

5.6.6.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.6.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.6.5 Dependency on Device State

None.

5.6.6.6 Effect on Device State

After expiration of specified time, *ConnectionStatus* is changed to *Disconnected* (via *PendingDisconnect* and *Disconnecting* state if it is implemented). The intermediate connection states before the connection is terminated will depend on *WarnDisconnectDelay*.

5.6.6.7 Errors

Table 20 — Error Codes for *SetAutoDisconnectTime()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.7 *SetIdleDisconnectTime()*

This action specifies the idle time (in seconds) after which a connection may be disconnected. The actual disconnect will occur after *WarnDisconnectDelay* time elapses. The process of terminating a connection is described in 5.7.2.

5.6.7.1 Arguments

Table 21 — Arguments for *SetIdleDisconnectTime()*

Argument	Direction	relatedStateVariable
NewIdleDisconnectTime	*IN*	*IdleDisconnectTime*

5.6.7.2 *NewIdleDisconnectTime*

This argument set the time of the connection idle before the connection is terminated. See definition in 5.4.8.

5.6.7.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.7.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.7.5 Dependency on Device State

None.

5.6.7.6 Effect on Device State

After expiration of specified time, *ConnectionStatus* is changed to *Disconnected* (via *PendingDisconnect* and *Disconnecting* state if it is implemented). The intermediate connection states before the connection is terminated will depend on *WarnDisconnectDelay*.

5.6.7.7 Errors

Table 22 — Error Codes for *SetIdleDisconnectTime()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.8 *SetWarnDisconnectDelay()*

This action specifies the number of seconds of warning to each (potentially) active user of a connection before a connection is terminated.

5.6.8.1 Arguments

Table 23 — Arguments for *SetWarnDisconnectDelay()*

Argument	Direction	relatedStateVariable
NewWarnDisconnectDelay	*IN*	*WarnDisconnectDelay*

5.6.8.2 *NewWarnDisconnectDelay*

This argument sets the time when IGD is expected to warn before a connection is terminated. See definition in 5.4.9.

5.6.8.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.8.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.8.5 Dependency on Device State

None.

5.6.8.6 Effect on Device State

After the time specified in seconds expires, the connection is terminated. *ConnectionStatus* is changed to *Disconnected* (via *Disconnecting* state if it is implemented).

5.6.8.7 Errors

Table 24 — Error Codes for *SetWarnDisconnectDelay()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.9 *GetStatusInfo()*

This action retrieves the values of state variables pertaining to connection status.

5.6.9.1 Arguments

Table 25 — Arguments for *GetStatusInfo()*

Argument	Direction	relatedStateVariable
NewConnectionStatus	*OUT*	*ConnectionStatus*
NewLastConnectionError	*OUT*	*LastConnectionError*
NewUptime	*OUT*	*Uptime*

5.6.9.2 *NewConnectionStatus*

This argument shows the current state of the connection. See definition in 5.4.4.

5.6.9.3 *NewLastConnectionError*

See explanation of this argument from 5.4.6.

5.6.9.4 *NewUptime*

See explanation of this argument from 5.4.5.

5.6.9.5 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.9.6 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.9.7 Dependency on Device State

None.

5.6.9.8 Effect on Device State

None.

5.6.9.9 Errors

Table 26 — Error Codes for *GetStatusInfo()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.10 *GetAutoDisconnectTime()*

This action retrieves the value of *AutoDisconnectTime* set by *SetAutoDisconnectTime()*. This value indicates the duration after the connection is terminated.

5.6.10.1 Arguments

Table 27 — Arguments for *GetAutoDisconnectTime()*

Argument	Direction	relatedStateVariable
NewAutoDisconnectTime	*OUT*	*AutoDisconnectTime*

5.6.10.2 *NewAutoDisconnectTime*

This argument defines duration in seconds after connection is closed automatically. See definition in 5.4.7.

5.6.10.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.10.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.10.5 Dependency on Device State

None.

5.6.10.6 Effect on Device State

None.

5.6.10.7 Errors

Table 28 — Error Codes for *GetAutoDisconnectTime()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.11 *GetIdleDisconnectTime()*

This action retrieves the value of *IdleDisconnectTime*. This value indicates how long a connection can stay in idle state before the connection termination.

5.6.11.1 Arguments

Table 29 — Arguments for *GetIdleDisconnectTime()*

Argument	Direction	relatedStateVariable
NewIdleDisconnectTime	*OUT*	*IdleDisconnectTime*

5.6.11.2 *NewIdleDisconnectTime*

This argument returns value of *IdleDisconnectTime* state variable defined in 5.4.8.

5.6.11.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.11.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.11.5 Dependency on Device State

None.

5.6.11.6 Effect on Device State

None.

5.6.11.7 Errors

Table 30 — Error Codes for *GetIdleDisconnectTime()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.12 *GetWarnDisconnectDelay()*

This action retrieves the values of *WarnDisconnectDelay*. This value indicates how long before a disconnection, the user is warned.

5.6.12.1 Arguments

Table 31 — Arguments for *GetWarnDisconnectDelay()*

Argument	Direction	relatedStateVariable
NewWarnDisconnectDelay	*OUT*	*WarnDisconnectDelay*

5.6.12.2 *NewWarnDisconnectDelay*

This argument returns the value of *WarnDisconnectDelay* state variable defined in 5.4.9.

5.6.12.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.12.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.12.5 Dependency on Device State

None.

5.6.12.6 Effect on Device State

None.

5.6.12.7 Errors

Table 32 — Error Codes for *GetWarnDisconnectDelay()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].

ErrorCode	errorDescription	Description
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.13 *GetNATRSIPStatus()*

This action retrieves the current state of NAT and RSIP on the gateway for this connection.

5.6.13.1 Arguments

Table 33 — Arguments for *GetNATRSIPStatus()*

Argument	Direction	relatedStateVariable
NewRSIPAvailable	*OUT*	*RSIPAvailable*
NewNATEnabled	*OUT*	*NATEnabled*

5.6.13.2 *NewRSIPAvailable*

This argument is type of **boolean** and describes if Realm Specific IP service has been enabled. See definition in 5.4.10.

5.6.13.3 *NewNATEnabled*

This argument is type of **boolean** and describes if NAT has been enabled. See definition in 5.4.11.

5.6.13.4 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.13.5 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.13.6 Dependency on Device State

None.

5.6.13.7 Effect on Device State

None.

5.6.13.8 Errors

Table 34 — Error Codes for *GetNATRSIPStatus()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.14 *GetGenericPortMappingEntry()*

This action retrieves NAT port mappings one entry at a time. Control points can call this action with an incrementing array index until no more entries are found on the gateway. If *PortMappingNumberOfEntries* is updated during a call, the process may have to start over. Entries in the array are contiguous. As entries are deleted, the array is compacted, and the evented variable *PortMappingNumberOfEntries* is decremented. Port mappings are logically

stored as an array on the IGD and retrieved using an array index ranging from 0 to *PortMappingNumberOfEntries*-1.

Returned port mappings rules correspond to those created by UPnP IGD control point, but also to those created by other mechanisms (ex: WWW-administration, remote management, local management…) shall be returned. In the meantime, NAT rules created through port triggering are not returned.

The returned port mappings also depends on the authentication of the control point (see 5.6.14.3).

5.6.14.1 Arguments

Table 35 — Arguments for *GetGenericPortMappingEntry()*

Argument	Direction	relatedStateVariable
NewPortMappingIndex	*IN*	*PortMappingNumberOfEntries*
NewRemoteHost	*OUT*	*RemoteHost*
NewExternalPort	*OUT*	*ExternalPort*
NewProtocol	*OUT*	*PortMappingProtocol*
NewInternalPort	*OUT*	*InternalPort*
NewInternalClient	*OUT*	*InternalClient*
NewEnabled	*OUT*	*PortMappingEnabled*
NewPortMappingDescription	*OUT*	*PortMappingDescription*
NewLeaseDuration	*OUT*	*PortMappingLeaseDuration*

See description of arguments from Table 11.

5.6.14.2 *NewPortMappingIndex*

This argument is the type of **ui2** and its allowed value range is from 0 to *PortMappingNumberOfEntries*-1. This parameter is an index to the table of all existing port mappings.

5.6.14.3 Service Requirements

This action returns an entry from the table of all port mapping entries based on *NewPortMappingIndex* argument.

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall not return the port mapping entry and shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

In particular, [3] recommends that unauthenticated and unauthorized control points are only allowed to retrieve port mapping entries which have:

- *InternalPort* and *ExternalPort* values greater than or equal to 1024,
- *InternalClient* value equals to the control point's IP address.

5.6.14.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.14.5 Dependency on Device State

None.

5.6.14.6 Effect on Device State

None.

5.6.14.7 Errors

Table 36 — Error Codes for *GetGenericPortMappingEntry()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
713	*SpecifiedArrayIndexInvalid*	The specified array index is out of bounds.

5.6.15 *GetSpecificPortMappingEntry()*

This action reports the port mapping specified by the unique tuple of *RemoteHost*, *ExternalPort* and *PortMappingProtocol*.

The returned port mappings also depends on the authentication of the control point (see 5.6.15.2).

5.6.15.1 Arguments

Table 37 — Arguments for *GetSpecificPortMappingEntry()*

Argument	Direction	relatedStateVariable
NewRemoteHost	*IN*	*RemoteHost*
NewExternalPort	*IN*	*ExternalPort*
NewProtocol	*IN*	*PortMappingProtocol*
NewInternalPort	*OUT*	*InternalPort*
NewInternalClient	*OUT*	*InternalClient*
NewEnabled	*OUT*	*PortMappingEnabled*
NewPortMappingDescription	*OUT*	*PortMappingDescription*
NewLeaseDuration	*OUT*	*PortMappingLeaseDuration*

See explanation of these arguments from Table 11.

5.6.15.2 Service Requirements

This action shall return the complete list of port mappings regardless of the creator. Returned port mappings rules correspond to those created by UPnP IGD control point, but also to those created by other mechanisms (ex: WWW-administration, remote management, local management...) shall be returned. In the meantime, NAT rules created through port triggerring are not returned.

If *NewRemoteHost* is equal to empty string value, *GetSpecificPortMappingEntry()* action shall return only port mappings with a *RemoteHost* equal to empty string value.

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall not return the port mapping entry and shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

In particular, [3] recommends that unauthenticated and unauthorized control points are only allowed to retrieve port mapping entries which have:

- *InternalPort* value greater than or equals to 1024,
- *InternalClient* value equals to the control point's IP address.

Moreover, [3] recommends that unauthenticated and unauthorized control points are only allowed to invoke this action with *NewExternalPort* value greater than or equals to 1024.

If the control point uses a *NewProtocol* value which is not in the allowedValueList of the *PortMappingProtocol* state variable, the gateway shall return the 601 *Argument Value Out of Range* error code (defined in [1]).

5.6.15.3 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.15.4 Dependency on Device State

None.

5.6.15.5 Effect on Device State

None.

5.6.15.6 Errors

Table 38 — Error Codes for *GetSpecificPortMappingEntry()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
714	*NoSuchEntryInArray*	The specified value does not exist in the array.

5.6.16 *AddPortMapping()*

Implementors should replace calls to *AddPortMapping()* with calls to *AddAnyPortMapping()*, effective with *WANIPConnection:2*.

This action creates a new port mapping or overwrites an existing mapping with the same internal client. If the *ExternalPort* and *PortMappingProtocol* pair is already mapped to another internal client, an error is returned.

When a control point creates a port forwarding rule with *AddPortMapping()* action for inbound traffic, this rule shall also be applied when NAT port triggering occurs for outbound traffic (see example in Figure 2).

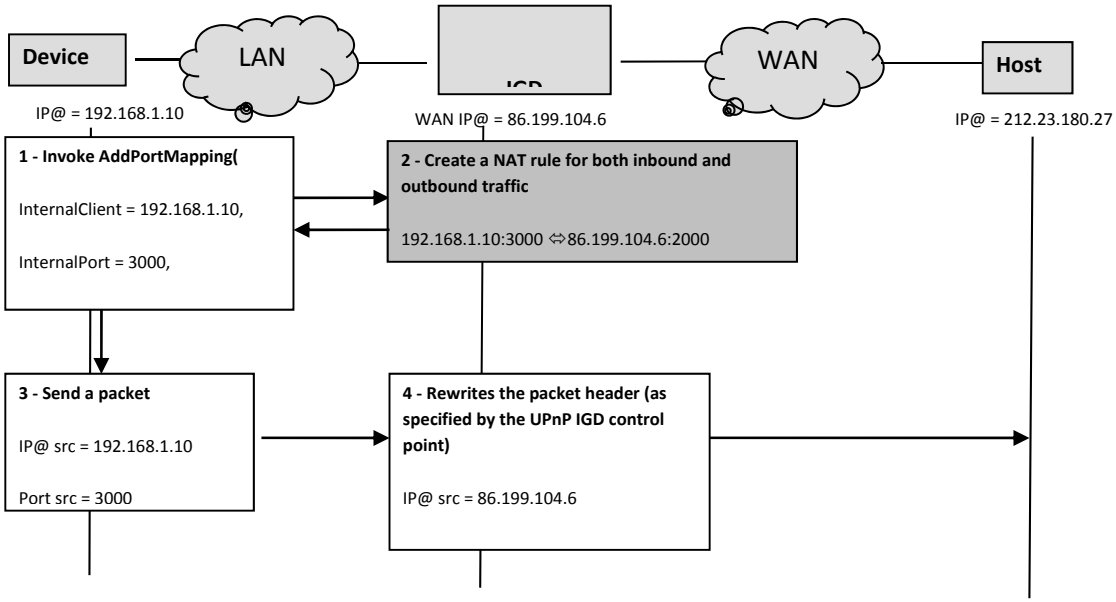

Figure 2 — Example of relationship between *AddPortMapping()* action and port triggering

Notice that: Not all NAT implementations will support:

- Wildcard value (i.e. 0) for *ExternalPort*,
- *InternalPort* values that are different from *ExternalPort*.

Deprecated error codes. Effective with *WANIPConnection:2*. These codes shall not be sent by *WANIPConnection*.

- 724: The gateway is required not to restrict port mappings to *ExternalPort* and *InternalPort* values being the same.
- 725: The gateway is required to support both wildcard and specific lease duration values for *PortMappingLeaseDuration*.
- 726: The gateway is required to support both wildcard and specific IP address values for *RemoteHost*.
- 727: The gateway is required to support both wildcard and specific port values for *ExternalPort*.

From this version, error code 724 MUST NOT be used, as the gateway is REQUIRED not to restrict port mappings to *ExternalPort* and *InternalPort* values being the same.

From this version, error code 725 MUST NOT be used as the gateway is REQUIRED to support both wildcard and specific lease duration values for *PortMappingLeaseDuration*.

From this version, error code 726 MUST NOT be used as the gateway is REQUIRED to support both wildcard and specific IP address values for *RemoteHost*.

From this version, error code 727 MUST NOT be used as the gateway is REQUIRED to support both wildcard and specific port values for *ExternalPort*.

5.6.16.1 Arguments

Table 39 — Arguments for *AddPortMapping()*

Argument	Direction	relatedStateVariable
NewRemoteHost	*IN*	*RemoteHost*
NewExternalPort	*IN*	*ExternalPort*
NewProtocol	*IN*	*PortMappingProtocol*

Argument	Direction	relatedStateVariable
NewInternalPort	*IN*	*InternalPort*
NewInternalClient	*IN*	*InternalClient*
NewEnabled	*IN*	*PortMappingEnabled*
NewPortMappingDescription	*IN*	*PortMappingDescription*
NewLeaseDuration	*IN*	*PortMappingLeaseDuration*

See description of these arguments from Table 11

5.6.16.2 Service Requirements

This action has restrictions on its arguments as follows:

- *NewPortMappingLeaseDuration* value can be between 1 second and 604800 seconds, however value “0” can exist for port mappings made out of band,
- Infinite mappings cannot be made in *WANIPConnection:2* and value “0” shall be interpreted as 604800 seconds,
- It is recommended that a lease time of 3600 seconds would be used as a default value.

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall not add the port mapping entry and shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

In particular, [3] recommends that unauthenticated and unauthorized control points are only allowed to invoke this action with:

- *NewExternalPort* and *NewInternalPort* values greater than or equal to 1024,
- *NewInternalClient* value equals to the control point's IP address.

If the control point uses a *NewProtocol* value which is not in the allowedValueList of the *PortMappingProtocol* state variable, the gateway shall return the 601 *Argument Value Out of Range* error code (defined in [1]).

In cases where the *ExternalPort*, *PortMappingProtocol* and *InternalClient* are the same, but *RemoteHost* is different, the vendor can choose to support both mappings simultaneously, or reject the second mapping with an appropriate error.

Protocol	ExternalPort	RemoteHost	InternalClient	Result
≠	≠	≠	≠	Success
≠	≠	≠	=	Success
≠	≠	=	≠	Success
≠	≠	=	=	Success
≠	=	≠	≠	Success
≠	=	≠	=	Success
≠	=	=	≠	Success
≠	=	=	=	Success
=	≠	≠	≠	Success
=	≠	≠	=	Success
=	≠	=	≠	Success
=	≠	=	=	Success

Protocol	ExternalPort	RemoteHost	InternalClient	Result
=	=	≠	≠	Failure
=	=	≠	=	Failure or success (vendor specific)
=	=	=	≠	Failure
=	=	=	=	Success (overwrite)

Figure 3 — Summary of *AddPortMapping()* results

5.6.16.3 Control Point Requirements When Calling The Action

When issuing *AddPortMapping()* action, control points are required to use following parameter values:

- *NewPortMappingLeaseDuration* value shall be between 1 second and 604800 seconds,
- It is recommended that *NewPortMappingLeaseDuration* value would be no more than 3600 seconds.

Before invoking this action, a control point should verify that it has sufficient permission.

Control points are required to support error codes 724, 725, 726 and 727 for legacy services, though *WANIPConnection:2* services shall not send those error codes.

5.6.16.4 Dependency on Device State

In order to have a successful port mapping request, the desired port mapping shall be free.

5.6.16.5 Effect on Device State

The effect is that a NAT portforwarding rule is set.

5.6.16.6 Errors

Table 40 — Error Codes for *AddPortMapping()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
715	*WildCardNotPermittedInSrcIP*	The source IP address cannot be wild-carded (*InternalClient* equals to an empty string).
716	*WildCardNotPermittedInExtPort*	The external port cannot be wild-carded.
718	*ConflictInMappingEntry*	The port mapping entry specified conflicts with a mapping assigned previously to another client.
724	*SamePortValuesRequired*	Deprecated Internal and External port values shall be the same. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.
725	*OnlyPermanentLeasesSupported*	Deprecated The NAT implementation only supports permanent lease times on port mappings. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.

ErrorCode	errorDescription	Description
726	*RemoteHostOnlySupportsWildcard*	Deprecated *RemoteHost* shall be a wildcard and cannot be a specific IP address or DNS name. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.
727	*ExternalPortOnlySupportsWildcard*	Deprecated *ExternalPort* shall be a wildcard and cannot be a specific port value. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.
728	*NoPortMapsAvailable*	There are not enough free ports available to complete port mapping.
729	*ConflictWithOtherMechanisms*	Attempted port mapping is not allowed due to conflict with other mechanisms.
732	*WildCardNotPermittedInIntPort*	The internal port cannot be wild-carded.

5.6.17 *AddAnyPortMapping()*

Like *AddPortMapping()* action, *AddAnyPortMapping()* action also creates a port mapping specified with the same arguments. The behaviour differs only on the case where the specified port is not free, because in that case the gateway reserves any free *NewExternalPort* and *NewProtocol* pair and returns the *NewReservedPort*. It is up to the vendors to define an algorithm which finds a free port.

It is encouraged to use this new action instead of the former one *AddPortMapping()* action, because it is more efficient, and it will be compatible with future potential solutions based on port range NAT solution also called "fractional address" within the IETF. The goal of "fractional address" NAT solution is to cope with the IPv4 public address exhaustion, by providing the same IPv4 public address to several IGDs, where each IGD is allocated with a different port range.

Notice that: Not all NAT implementations will support:

- Wildcard value (i.e. 0) for *ExternalPort*,
- *InternalPort* values that are different from *ExternalPort*.

Regarding the last point, this behaviour is not encouraged because the goal of *AddAnyPortMapping()* is to provide a free port if the desired port is not free, so the *InternalPort* is potentially different from the *ExternalPort*. Nevertheless, in order to be backward compatible with *AddPortMapping()* action, this behaviour is supported. If parameters *NewInternalClient*, *NewExternalPort* and *NewProtocol* are the same as an existing port mapping and control point is authorized for the operation, the port mapping is updated instead of creating new one.

When a control point creates a port forwarding rule with *AddAnyPortMapping()* for inbound traffic, this rule shall also be applied when NAT port triggering occurs for outbound traffic (see example in Figure 2).

5.6.17.1 Arguments

Table 41 — Arguments for *AddAnyPortMapping()*

Argument	Direction	relatedStateVariable
NewRemoteHost	*IN*	*RemoteHost*
NewExternalPort	*IN*	*ExternalPort*
NewProtocol	*IN*	*PortMappingProtocol*
NewInternalPort	*IN*	*InternalPort*
NewInternalClient	*IN*	*InternalClient*
NewEnabled	*IN*	*PortMappingEnabled*

Argument	Direction	relatedStateVariable
NewPortMappingDescription	*IN*	*PortMappingDescription*
NewLeaseDuration	*IN*	*PortMappingLeaseDuration*
NewReservedPort	*OUT*	*ExternalPort*

See explanation of arguments from Table 11 except *NewReservedPort*.

5.6.17.2 *NewReservedPort*

This argument's type is **ui2** and is used to describe the number of the port reserved.

5.6.17.3 Service Requirements

This action has restrictions on its arguments as follows:

- *NewPortMappingLeaseDuration* value can be between 1 second and 604800 seconds, however value "0" can exist for port mappings made out of band,
- Static mappings cannot be made in *WANIPConnection:2*. Value "0" shall be interpreted as 604800 seconds,
- It is recommended that a lease time of 3600 seconds be used as a default value,
- If *NewExternalPort* is specified as a wildcard (i.e. 0) and if the gateway supports that feature, the action will return 0 in the *NewReservedPort* parameter.

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall not add the port mapping entry and shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

In particular, [3] recommends that unauthenticated and unauthorized control points are only allowed to invoke this action with:

- *NewExternalPort* and *NewInternalPort* values greater than or equal to 1024,
- *NewInternalClient* value equals to the control point's IP address.

Moreover, [3] recommends that unauthenticated and unauthorized control points are only allowed to create port mapping entries which have *ExternalPort* value greater than or equals to 1024.

If the control point uses a *NewProtocol* value which is not in the allowedValueList of the *PortMappingProtocol* state variable, the gateway shall return the 601 *Argument Value Out of Range* error code (defined in [1]).

In cases where the *ExternalPort*, *PortMappingProtocol* and *InternalClient* are the same, but *RemoteHost* is different, the vendor can choose to support both mappings simultaneously, or to create a new one (with an alternative *ExternalPort*) for the second mapping.

Protocol	ExternalPort	RemoteHost	InternalClient	Result
≠	≠	≠	≠	Success (ReservedPort = ExternalPort)
≠	≠	≠	=	Success (ReservedPort = ExternalPort)
≠	≠	=	≠	Success (ReservedPort = ExternalPort)
≠	≠	=	=	Success (ReservedPort = ExternalPort)
≠	=	≠	≠	Success (ReservedPort = ExternalPort)
≠	=	≠	=	Success (ReservedPort = ExternalPort)
≠	=	=	≠	Success (ReservedPort = ExternalPort)

Protocol	ExternalPort	RemoteHost	InternalClient	Result
≠	=	=	=	Success (ReservedPort = ExternalPort)
=	≠	≠	≠	Success (ReservedPort = ExternalPort)
=	≠	≠	=	Success (ReservedPort = ExternalPort)
=	≠	=	≠	Success (ReservedPort = ExternalPort)
=	≠	=	=	Success (ReservedPort = ExternalPort)
=	=	≠	≠	Success (ReservedPort ≠ ExternalPort)
=	=	≠	=	Success (vendor specific either ReservedPort ≠ ExternalPort or ReservedPort = ExternalPort)
=	=	=	≠	Success (ReservedPort ≠ ExternalPort)
=	=	=	=	Success (overwrite)

Figure 4 — Summary of *AddAnyPortMapping()* results

5.6.17.4 Control Point Requirements When Calling The Action

When issuing *AddAnyPortMapping()* action, control points are required to use the following parameter values:

- *NewLeaseDuration* value shall be between 1 second and 604800 seconds,
- It is recommended that *NewLeaseDuration* value be no more than 3600 seconds.

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.17.5 Dependency on Device State

In order to have a successful port mapping request, there shall be free port mappings available.

5.6.17.6 Effect on Device State

The effect is that a NAT portforwarding rule is set.

5.6.17.7 Errors

Table 42 — Error Codes for *AddAnyPortMapping()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
715	*WildCardNotPermittedInSrcIP*	The source IP address cannot be wild-carded (*InternalClient* equals to an empty string).
716	*WildCardNotPermittedInExtPort*	The external port cannot be wild-carded.
728	*NoPortMapsAvailable*	There are not enough free ports available to complete port mapping.
729	*ConflictWithOtherMechanisms*	Attempted port mapping is not allowed due to conflict with other mechanisms.
732	*WildCardNotPermittedInIntPort*	The internal port cannot be wild-carded.

5.6.18 *DeletePortMapping()*

This action deletes a previously instantiated port mapping. As each entry is deleted, the array is compacted, the evented variable *PortMappingNumberOfEntries* is decremented and the evented variable *SystemUpdateID* is incremented.

5.6.18.1 Arguments

Table 43 — Arguments for *DeletePortMapping()*

Argument	Direction	relatedStateVariable
NewRemoteHost	*IN*	*RemoteHost*
NewExternalPort	*IN*	*ExternalPort*
NewProtocol	*IN*	*PortMappingProtocol*

See description from Table 11.

5.6.18.2 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall not delete the port mapping entry and shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

In particular, [3] recommends that unauthenticated and unauthorized control points are only allowed to delete port mapping entries which have:

- *InternalPort* value greater than or equals to 1024,
- *InternalClient* value equals to the control point's IP address.

Moreover, [3] recommends that unauthenticated and unauthorized control points are only allowed to invoke this action with *NewExternalPort* value greater than or equals to 1024.

If the control point uses a *NewProtocol* value which is not in the allowedValueList of the *PortMappingProtocol* state variable, the gateway shall return the 601 *Argument Value Out of Range* error code (defined in [1]).

5.6.18.3 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.18.4 Dependency on Device State

There needs to be existing port mapping entry in order to this action to succeed.

5.6.18.5 Effect on Device State

Inbound connections are no longer permitted on the port mapping being deleted.

5.6.18.6 Errors

Table 44 — Error Codes for *DeletePortMapping()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
714	*NoSuchEntryInArray*	The specified value does not exist in the array.

5.6.19 *DeletePortMappingRange()*

This action deletes port mapping entries defined by a range. As the range is deleted, the array is compacted, the evented variable *PortMappingNumberOfEntries* is decremented, and the evented variable *SystemUpdateID* is incremented. When issued, this action will remove all port mapping entries between *NewStartPort* and *NewEndPort*.

The *NewManage* argument is used to describe the intent of this action:

- If *NewManage* is set to "**0**" (false), then the gateway shall only remove port mappings having the *InternalClient* value matching IP address of the control point,
- If *NewManage* is set to "**1**" (true), the gateway shall remove all port mappings between *NewStartPort* and *NewEndPort* values.

The deleted port mappings also depends on the authentication of the control point (see 5.6.19.2).

5.6.19.1 Arguments

Table 45 — Arguments for *DeletePortMappingRange()*

Argument	Direction	relatedStateVariable
NewStartPort	*IN*	*ExternalPort*
NewEndPort	*IN*	*ExternalPort*
NewProtocol	*IN*	*PortMappingProtocol*
NewManage	*IN*	*A_ARG_TYPE_Manage*

See description of arguments from Table 11.

5.6.19.2 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]). However, if the control point does not have the necessary permission to delete a port mapping entry in the specified range, the service should skip it and check next port mapping entries in the range. Finally, if no port mapping entries are found in the specified range, the service shall return the 730 *PortMappingNotFound* error code.

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

In particular, [3] recommends that unauthenticated and unauthorized control points are only allowed to delete port mapping entries which have:

- *InternalPort* value greater than or equals to 1024,
- *InternalClient* value equals to the control point's IP address.

Moreover, [3] recommends that unauthenticated and unauthorized control points are only allowed to invoke this action with *NewStartPort* and *NewEndPort* values greater than or equal to 1024.

If the control point is not authenticated, the *NewManage* argument should be ignored.

If the control point uses a *NewProtocol* value which is not in the allowedValueList of the *PortMappingProtocol* state variable, the gateway shall return the 601 *Argument Value Out of Range* error code (defined in [1]).

It is advised that when using this action, application developers should take caution not to inadvertently destroy port mappings made by other control points from the same IP address.

This action shall be done as atomic operation where all deletable port mappings are deleted according the given criteria.

5.6.19.3 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.19.4 Dependency on Device State

There needs to be existing at least one port mapping entry in the range in order to this action succeeds.

5.6.19.5 Effect on Device State

Inbound connections are no longer permitted on the port mapping being deleted.

5.6.19.6 Errors

Table 46 — Error Codes for *DeletePortMappingRange()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
730	*PortMappingNotFound*	This error message is returned if no port mapping is found in the specified range.
733	*InconsistentParameters*	*NewStartPort* and *NewEndPort* values are not consistent.

5.6.20 *GetExternalIPAddress()*

This action retrieves the value of the external IP address on this connection instance.

5.6.20.1 Arguments

Table 47 — Arguments for *GetExternalIPAddress()*

Argument	Direction	relatedStateVariable
NewExternalIPAddress	*OUT*	*ExternalIPAddress*

5.6.20.2 Argument Descriptions

See description in Table 11.

5.6.20.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]).

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

5.6.20.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that is has sufficient permission.

5.6.20.5 Dependency on Device State

None.

5.6.20.6 Effect on Device State

None.

5.6.20.7 Errors

Table 48 — Error Codes for *GetExternalIPAddress()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized.

5.6.21 *GetListOfPortMappings()*

This action returns a list of port mappings matching the arguments. The operation of this action has two modes depending on *NewManage* value:

- If the *NewManage* argument is set to "**0**" (false), then this action returns a list of port mappings that have *InternalClient* value matching to the IP address of the control point between *NewStartPort* and *NewEndPort*,
- If the *NewManage* argument is set to "**1**" (true), then the gateway shall return all port mappings between *NewStartPort* and *NewEndPort*.

With the argument *NewNumberOfPorts*, a control point may limit the size of the list returned in order to limit the length of the list returned. If *NewNumberOfPorts* is equal to 0, then the gateway shall return all port mappings between *NewStartPort* and *NewEndPort*.

The returned port mappings also depends on the authentication of the control point (see 5.6.21.3).

5.6.21.1 Arguments

Table 49 — Arguments for *GetListOfPortMappings()*

Argument	Direction	relatedStateVariable
NewStartPort	*IN*	*ExternalPort*
NewEndPort	*IN*	*ExternalPort*
NewProtocol	*IN*	*PortMappingProtocol*
NewManage	*IN*	*A_ARG_TYPE_Manage*
NewNumberOfPorts	*IN*	*PortMappingNumberOfEntries*
NewPortListing	*OUT*	*A_ARG_TYPE_PortListing*

See descriptions for *NewStartPort* and *NewEndPort* in Table 11.

5.6.21.2 *NewPortListing*

This argument returns a list of port mappings matching the input arguments. This is a XML fragment defined in *A_ARG_TYPE_PortListing* state variable description.

5.6.21.3 Service Requirements

Before processing the action request, the service shall apply the chosen security policy, and authenticate and authorize the control point as required by the security policy. If the control point does not have the necessary permission to perform the action with the requested parameters, the service shall return the 606 *Action not authorized* error code (defined in [1]). However, if the control point does not have the necessary permission to retrieve a specific port mapping entry in the specied range, the service should skip it and check next port mapping entries in the range. Finally, if no port mapping entries are found in the specified range, the service shall return the 730 *PortMappingNotFound* error code.

[3] recommends access control requirements and authentication levels to be applied by *default* for this action. However, services may choose a different security policy.

In particular, [3] recommends that unauthenticated and unauthorized control points are only allowed to invoke this action with *NewStartPort* and *NewEndPort* values greater than or equal to 1024.

Moreover, [3] recommends that unauthenticated and unauthorized control points are only allowed to retrieve port mapping entries which have:

- *InternalPort* value greater than or equals to 1024,
- *InternalClient* value equals to the control point's IP address.

If the control point is not authenticated, the *NewManage* argument should be ignored.

If the control point uses a *NewProtocol* value which is not in the allowedValueList of the *PortMappingProtocol* state variable, the gateway shall return the 601 *Argument Value Out of Range* error code (defined in [1]).

5.6.21.4 Control Point Requirements When Calling The Action

Before invoking this action, a control point should verify that it has sufficient permission.

5.6.21.5 Dependency on Device State

None.

5.6.21.6 Effect on Device State

None.

5.6.21.7 Errors

Table 50 — Error Codes for *GetListOfPortMappings()*

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
730	*PortMappingNotFound*	This error message is returned if no port mapping is found in the specified range.
733	*InconsistentParameters*	*NewStartPort* and *NewEndPort* values are not consistent.

5.6.22 Relationships Between Actions

Actions initiated by a client may have different results depending on whether the state of the gateway was changed as a result of another client's actions.

For example, the action *RequestConnection()* might not be successful in changing the *ConnectionStatus* to *Connected* if the gateway receives *RequestTermination()* on the same connection (while it is in the process of connecting) from another client.

5.6.23 Error Code Summary

The Table 51 lists error codes common to actions for this service type. If an action results in multiple errors, the most specific error should be returned.

Table 51 — Error Code Summary

ErrorCode	errorDescription	Description
400-499	TBD	See Control clause in [1].
500-599	TBD	See Control clause in [1].
600-699	TBD	See Control clause in [1].
606	*Action not authorized*	The action requested requires authorization and the sender was not authorized. This error can be returned e.g. if a control point is not authorized for the operation because of its IP address or because it tries to use well-known ports.
700		Reserved for future extensions.
703	*InactiveConnectionStateRequired*	Current value of *ConnectionStatus* should be either *Disconnected* or *Unconfigured* to permit this action.
704	*ConnectionSetupFailed*	There was a failure in setting up the IP or PPP connection with the service provider.
705	*ConnectionSetupInProgress*	The connection is already in the process of being setup.
706	*ConnectionNotConfigured*	Current *ConnectionStatus* is *Unconfigured*.
707	*DisconnectInProgress*	The connection is in the process of being torn down.
708	*InvalidLayer2Address*	Corresponding Link Config service has an invalid VPI/VCI or phone number.
709	*InternetAccessDisabled*	The *EnabledForInternet* flag is set to "**0**" (false).
710	*InvalidConnectionType*	This action is not permitted for the specified *ConnectionType*.
711	*ConnectionAlreadyTerminated*	An attempt was made to terminate a connection that is no longer active.
713	*SpecifiedArrayIndexInvalid*	The specified array index is out of bounds.
714	*NoSuchEntryInArray*	The specified value does not exist in the array.
715	*WildCardNotPermittedInSrcIP*	The source IP address cannot be wild-carded.
716	*WildCardNotPermittedInExtPort*	The external port cannot be wild-carded.
718	*ConflictInMappingEntry*	The port mapping entry specified conflicts with a mapping assigned previously to another client.
724	*SamePortValuesRequired*	Deprecated Internal and External port values shall be the same. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.
725	*OnlyPermanentLeasesSupported*	Deprecated The NAT implementation only supports permanent lease times on port mappings. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.
726	*RemoteHostOnlySupportsWildcard*	Deprecated *RemoteHost* shall be a wildcard and cannot be a specific IP address or DNS name. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.
727	*ExternalPortOnlySupportsWildcard*	Deprecated *ExternalPort* shall be a wildcard and cannot be a specific port value. This error code shall not be used by *WANIPConnection:2* services, but Control points are required to support this error code.
728	*NoPortMapsAvailable*	There are not enough free ports available to complete port mapping.
729	*ConflictWithOtherMechanisms*	Attempted port mapping is not allowed due to conflict with other mechanisms.
730	*PortMappingNotFound*	This error message is returned if no port mapping is found in the specified range.

ErrorCode	errorDescription	Description
731	*ReadOnly*	Not possible to modify the value because it is read only.
732	*WildCardNotPermittedInIntPort*	The internal port cannot be wild-carded.
733	*InconsistentParameters*	*NewStartPort* and *NewEndPort* values are not consistent.

NOTE: 800-899 Error Codes are not permitted for standard actions. See Control clause in [1] for more details.

5.7 Service Behavioral Model

5.7.1 Connection Initiation

When a *WANConnectionDevice* is initialized, an instance of *WANIPConnection* service will be initialized. If an IP connection is automatically initiated i.e. 'always on' as soon as the underlying link is up, no action is needed from a UPnP control point to initiate the connection. However, the IP connection may become inactive (*Disconnected*) because of network or server issues.

A UPnP client sends the *RequestConnection()* action to a specific instance of the *WANIPConnection* service on a particular *WANConnectionDevice* to inform the gateway of its intent to use Internet access.

When a client sends a *RequestConnection()* command to a *Disconnected* connection, the *WANConnectionDevice* initiates the connection to ISP and may set *ConnectionStatus* to *Connecting*. Depending on whether the connection is successful, *ConnectionStatus* is changed to *Connected* or *Disconnected*.

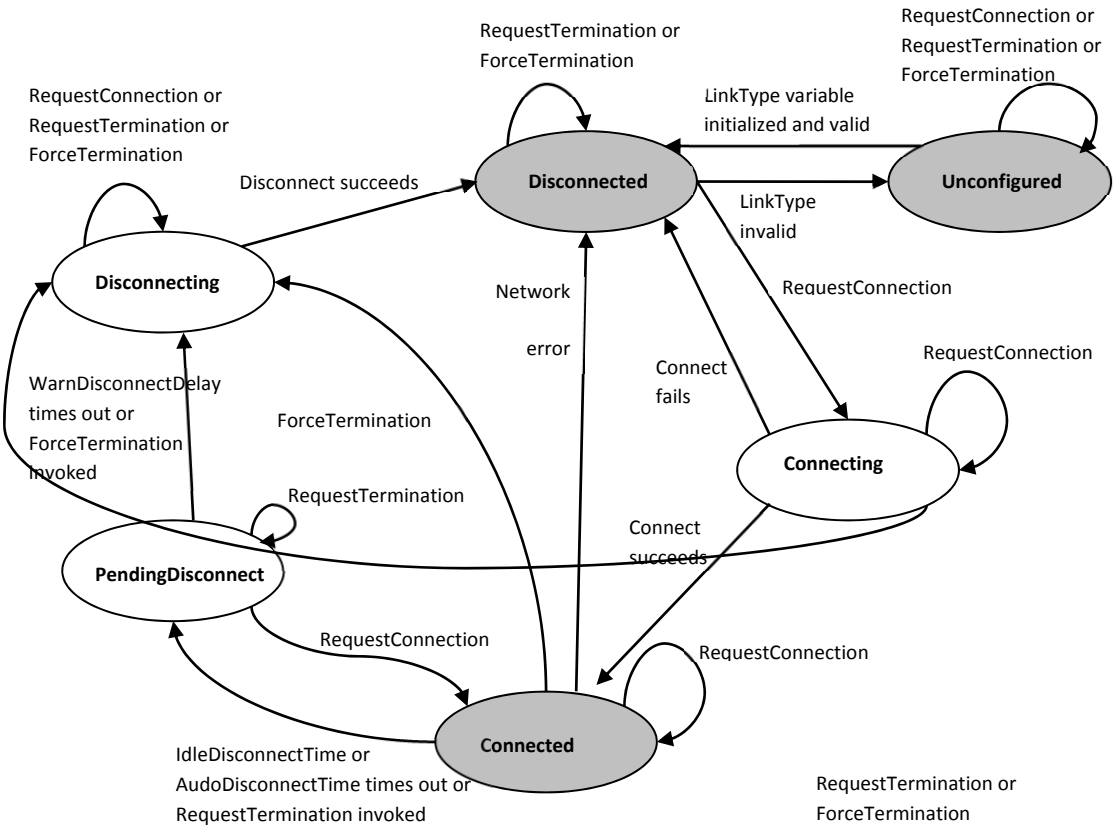

Figure 5 — State diagram for IP connection

When a connection service gets a *RequestConnection()* command, if the *ConnectionStatus* is:

- *Unconfigured*, *Connecting* or *Disconnecting*: an error is returned,
- *Disconnected*: a connection is attempted (*ConnectionStatus* may transition to *Connecting*). If this is successful, *ConnectionStatus* changes to *Connected*,
- *PendingDisconnect*: it is changed to *Connected*.

When the *ConnectionStatus* is *Connected*: the client is allowed to use the connection if *ConnectionType* is *IP_Routed*, otherwise an error is returned.

Figure 5 illustrates the state transition diagram when all states are implemented by the gateway. Required states are in shaded ovals.

RequestConnection() may fail (causing an error code to be returned) under the following conditions:

d) Network failure,

e) *ConnectionStatus* is *Unconfigured*, *Connecting* or *Disconnecting*,

f) *EnabledForInternet* variable in *WANCommonInterfaceConfig* is set to "**0**" (false),

g) Invalid *ConnectionType* (either *Unconfigured* or *IP_Bridged*).

The connection set up may be aborted by a client (by issuing *RequestTermination()* or *ForceTermination()*).

5.7.2 Connection Termination

Connection termination can be explicit (by a client sending *RequestTermination()* or *ForceTermination()* action) or implicit (because of *AutoDisconnectTime* or *IdleDisconnectTime* coming into effect).

A UPnP client sends *RequestTermination()* or *ForceTermination()* action to a specific instance of the *WANIPConnection* service on a particular *WANConnectionDevice* to inform the gateway that this client no longer needs IP services.

A *RequestTermination()* command is acted upon only if the *ConnectionType* is *IP_Routed* and *ConnectionStatus* is *Connecting* or *Connected*, while a *ForceTermination()* command is acted upon only if the *ConnectionType* is *IP_Routed* and *ConnectionStatus* is *Connected*, *Connecting* or *PendingDisconnect*.

A connection termination may be initiated due to:

h) A *RequestTermination()* or *ForceTermination()* command from a client,

i) *AutoDisconnectTime* or *IdleDisconnectTime* coming into effect,

j) A deployment specific gateway policy,

k) *EnabledForInternet* variable (in *WANCommonInterfaceConfig* service) being set to "**0**" (false),

l) An ISP initiated connection termination or network failure.

At this point *ConnectionStatus* transitions (resulting in notification to clients registered for this event) immediately to one of the following:

- *PendingDisconnect* (if this state is implemented): This occurs if *WarnDisconnectDelay* is non-zero and the cause for termination is 5.7.2h) or 5.7.2i)—as mentioned above. The IP connection is still active in this state. This is useful for giving clients using a connection a chance to react when a connection termination is in progress. If the termination is due to a gateway policy—5.7.2j) above—a specific implementation of the gateway may choose to warn the clients by transitioning to this state.
  - If clients choose to ignore the notification, the connection will be terminated after the time (in seconds) specified as *WarnDisconnectDelay*. *ConnectionStatus* transitions to *Disconnecting*.

- If any client sends *RequestConnection()* command at this point, the gateway may choose to discontinue the termination process by changing *ConnectionStatus* to *Connected*. If connection is not restored, the gateway will return error code indicating that the connection was in the process of being torn down.

- *Disconnecting* – this can happen in the following cases –
  - *ForceTermination()* command was called,
  - *RequestTermination()* called, and if no other clients are using the connection, the gateway may choose to skip *PendingDisconnect* state,
  - *WarnDisconnectDelay* is zero and the cause for termination is *RequestTermination()* or 2 (as mentioned above),
  - Termination was triggered by *EnabledForInternet* variable being set to "**0**" (false),
  - Termination was triggered by ISP,
  - Termination occurred due to a gateway policy, and the specific implementation chose not to warn the clients by switching directly to this state – essentially overriding the value of *WarnDisconnectDelay*.
- *Disconnected* – if the above two allowed states are not implemented.

When transitioning to this state, the connection is terminated immediately.

If the connection state is *Connecting* when a client issues a *RequestTermination()*, the state transitions to *Disconnecting* directly – it does not go to *PendingDisconnect* even if *WarnDisconnectDelay* is nonzero.

As mentioned before, in the case of termination because of a gateway policy the action (whether clients are warned or not) depends upon the gateway implementation.

When a client receives a *PendingDisconnect* notification, it can do one of two things:

- Ignore it and let the disconnect proceed,
- Send a *RequestConnection()* command – the client can keep the connection from disconnecting – this is implementation dependent as pointed out earlier.

6 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>SetConnectionType</name>
         <argumentList>
            <argument>
               <name>NewConnectionType</name>
               <direction>in</direction>
               <relatedStateVariable>
                  ConnectionType
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>GetConnectionTypeInfo</name>
         <argumentList>
            <argument>
```

```
            <name>NewConnectionType</name>
            <direction>out</direction>
            <relatedStateVariable>
               ConnectionType
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewPossibleConnectionTypes</name>
            <direction>out</direction>
            <relatedStateVariable>
               PossibleConnectionTypes
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>RequestConnection</name>
   </action>

   <action>
      <name>RequestTermination</name>
   </action>

   <action>
      <name>ForceTermination</name>
   </action>

   <action>
      <name>SetAutoDisconnectTime</name>
      <argumentList>
         <argument>
            <name>NewAutoDisconnectTime</name>
            <direction>in</direction>
            <relatedStateVariable>
               AutoDisconnectTime
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>SetIdleDisconnectTime</name>
      <argumentList>
         <argument>
            <name>NewIdleDisconnectTime</name>
            <direction>in</direction>
            <relatedStateVariable>
               IdleDisconnectTime
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>SetWarnDisconnectDelay</name>
      <argumentList>
         <argument>
            <name>NewWarnDisconnectDelay</name>
            <direction>in</direction>
            <relatedStateVariable>
               WarnDisconnectDelay
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
```



```
      <name>GetStatusInfo</name>
      <argumentList>
         <argument>
            <name>NewConnectionStatus</name>
            <direction>out</direction>
            <relatedStateVariable>
               ConnectionStatus
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewLastConnectionError</name>
            <direction>out</direction>
            <relatedStateVariable>
               LastConnectionError
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewUptime</name>
            <direction>out</direction>
            <relatedStateVariable>
               Uptime
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>GetAutoDisconnectTime</name>
      <argumentList>
         <argument>
            <name>NewAutoDisconnectTime</name>
            <direction>out</direction>
            <relatedStateVariable>
               AutoDisconnectTime
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>GetIdleDisconnectTime</name>
      <argumentList>
         <argument>
            <name>NewIdleDisconnectTime</name>
            <direction>out</direction>
            <relatedStateVariable>
               IdleDisconnectTime
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>GetWarnDisconnectDelay</name>
      <argumentList>
         <argument>
            <name>NewWarnDisconnectDelay</name>
            <direction>out</direction>
            <relatedStateVariable>
               WarnDisconnectDelay
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>GetNATRSIPStatus</name>
```

```
            <argumentList>
               <argument>
                  <name>NewRSIPAvailable</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     RSIPAvailable
                  </relatedStateVariable>
               </argument>

               <argument>
                  <name>NewNATEnabled</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     NATEnabled
                  </relatedStateVariable>
               </argument>
            </argumentList>
         </action>

         <action>
            <name>GetGenericPortMappingEntry</name>
            <argumentList>
               <argument>
                  <name>NewPortMappingIndex</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     PortMappingNumberOfEntries
                  </relatedStateVariable>
               </argument>

               <argument>
                  <name>NewRemoteHost</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     RemoteHost
                  </relatedStateVariable>
               </argument>

               <argument>
                  <name>NewExternalPort</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     ExternalPort
                  </relatedStateVariable>
               </argument>

               <argument>
                  <name>NewProtocol</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     PortMappingProtocol
                  </relatedStateVariable>
               </argument>

               <argument>
                  <name>NewInternalPort</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     InternalPort
                  </relatedStateVariable>
               </argument>

               <argument>
                  <name>NewInternalClient</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     InternalClient
                  </relatedStateVariable>
               </argument>
```



```
            <argument>
               <name>NewEnabled</name>
               <direction>out</direction>
               <relatedStateVariable>
                  PortMappingEnabled
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewPortMappingDescription</name>
               <direction>out</direction>
               <relatedStateVariable>
                  PortMappingDescription
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewLeaseDuration</name>
               <direction>out</direction>
               <relatedStateVariable>
                  PortMappingLeaseDuration
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>GetSpecificPortMappingEntry</name>
         <argumentList>
            <argument>
               <name>NewRemoteHost</name>
               <direction>in</direction>
               <relatedStateVariable>
                  RemoteHost
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewExternalPort</name>
               <direction>in</direction>
               <relatedStateVariable>
                  ExternalPort
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewProtocol</name>
               <direction>in</direction>
               <relatedStateVariable>
                  PortMappingProtocol
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewInternalPort</name>
               <direction>out</direction>
               <relatedStateVariable>
                  InternalPort
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewInternalClient</name>
               <direction>out</direction>
               <relatedStateVariable>
                  InternalClient
               </relatedStateVariable>
            </argument>
```

```
            <argument>
               <name>NewEnabled</name>
               <direction>out</direction>
               <relatedStateVariable>
                  PortMappingEnabled
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewPortMappingDescription</name>
               <direction>out</direction>
               <relatedStateVariable>
                  PortMappingDescription
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewLeaseDuration</name>
               <direction>out</direction>
               <relatedStateVariable>
                  PortMappingLeaseDuration
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>AddPortMapping</name>
         <argumentList>
            <argument>
               <name>NewRemoteHost</name>
               <direction>in</direction>
               <relatedStateVariable>
                  RemoteHost
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewExternalPort</name>
               <direction>in</direction>
               <relatedStateVariable>
                  ExternalPort
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewProtocol</name>
               <direction>in</direction>
               <relatedStateVariable>
                  PortMappingProtocol
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewInternalPort</name>
               <direction>in</direction>
               <relatedStateVariable>
                  InternalPort
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewInternalClient</name>
               <direction>in</direction>
               <relatedStateVariable>
                  InternalClient
               </relatedStateVariable>
            </argument>
```



```
            <argument>
               <name>NewEnabled</name>
               <direction>in</direction>
               <relatedStateVariable>
                  PortMappingEnabled
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewPortMappingDescription</name>
               <direction>in</direction>
               <relatedStateVariable>
                  PortMappingDescription
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewLeaseDuration</name>
               <direction>in</direction>
               <relatedStateVariable>
                  PortMappingLeaseDuration
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>DeletePortMapping</name>
         <argumentList>
            <argument>
               <name>NewRemoteHost</name>
               <direction>in</direction>
               <relatedStateVariable>
                  RemoteHost
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewExternalPort</name>
               <direction>in</direction>
               <relatedStateVariable>
                  ExternalPort
               </relatedStateVariable>
            </argument>

            <argument>
               <name>NewProtocol</name>
               <direction>in</direction>
               <relatedStateVariable>
                  PortMappingProtocol
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>GetExternalIPAddress</name>
         <argumentList>
            <argument>
               <name>NewExternalIPAddress</name>
               <direction>out</direction>
               <relatedStateVariable>
                  ExternalIPAddress
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
```

```
<action>
   <name>DeletePortMappingRange</name>
   <argumentList>
      <argument>
         <name>NewStartPort</name>
         <direction>in</direction>
         <relatedStateVariable>
            ExternalPort
         </relatedStateVariable>
      </argument>

      <argument>
         <name>NewEndPort</name>
         <direction>in</direction>
         <relatedStateVariable>
            ExternalPort
         </relatedStateVariable>
      </argument>

      <argument>
         <name>NewProtocol</name>
         <direction>in</direction>
         <relatedStateVariable>
            PortMappingProtocol
         </relatedStateVariable>
      </argument>

      <argument>
         <name>NewManage</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_Manage
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>

<action>
   <name>GetListOfPortMappings</name>
   <argumentList>
      <argument>
         <name>NewStartPort</name>
         <direction>in</direction>
         <relatedStateVariable>
            ExternalPort
         </relatedStateVariable>
      </argument>

      <argument>
         <name>NewEndPort</name>
         <direction>in</direction>
         <relatedStateVariable>
            ExternalPort
         </relatedStateVariable>
      </argument>

      <argument>
         <name>NewProtocol</name>
         <direction>in</direction>
         <relatedStateVariable>
            PortMappingProtocol
         </relatedStateVariable>
      </argument>

      <argument>
         <name>NewManage</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_Manage
```



```
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewNumberOfPorts</name>
            <direction>in</direction>
            <relatedStateVariable>
               PortMappingNumberOfEntries
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewPortListing</name>
            <direction>out</direction>
            <relatedStateVariable>
               A_ARG_TYPE_PortListing
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

   <action>
      <name>AddAnyPortMapping</name>
      <argumentList>
         <argument>
            <name>NewRemoteHost</name>
            <direction>in</direction>
            <relatedStateVariable>
               RemoteHost
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewExternalPort</name>
            <direction>in</direction>
            <relatedStateVariable>
               ExternalPort
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewProtocol</name>
            <direction>in</direction>
            <relatedStateVariable>
               PortMappingProtocol
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewInternalPort</name>
            <direction>in</direction>
            <relatedStateVariable>
               InternalPort
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewInternalClient</name>
            <direction>in</direction>
            <relatedStateVariable>
               InternalClient
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewEnabled</name>
            <direction>in</direction>
            <relatedStateVariable>
               PortMappingEnabled
```

```
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewPortMappingDescription</name>
            <direction>in</direction>
            <relatedStateVariable>
               PortMappingDescription
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewLeaseDuration</name>
            <direction>in</direction>
            <relatedStateVariable>
               PortMappingLeaseDuration
            </relatedStateVariable>
         </argument>

         <argument>
            <name>NewReservedPort</name>
            <direction>out</direction>
            <relatedStateVariable>
               ExternalPort
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

    <!-- Declarations for other actions defined by UPnP vendor (if any) go here -->

</actionList>

<serviceStateTable>

   <stateVariable sendEvents="no">
      <name>ConnectionType</name>
      <dataType>string</dataType>
      <defaultValue>IP_Routed</defaultValue>
      <allowedValueList>
         <allowedValue>Unconfigured</allowedValue>
         <allowedValue>IP_Routed</allowedValue>
         <allowedValue>IP_Bridged</allowedValue>
      </allowedValueList>
   </stateVariable>

   <stateVariable sendEvents="yes">
      <name>PossibleConnectionTypes</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="yes">
      <name>ConnectionStatus</name>
      <dataType>string</dataType>
      <allowedValueList>
         <allowedValue>Unconfigured</allowedValue>
         <allowedValue>Connecting</allowedValue>
         <allowedValue>Connected</allowedValue>
         <allowedValue>PendingDisconnect</allowedValue>
         <allowedValue>Disconnecting</allowedValue>
         <allowedValue>Disconnected</allowedValue>
      </allowedValueList>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>Uptime</name>
      <dataType>ui4</dataType>
   </stateVariable>
```



```
<stateVariable sendEvents="no">
   <name>LastConnectionError</name>
   <dataType>string</dataType>
   <allowedValueList>
      <allowedValue>ERROR_NONE</allowedValue>
      <allowedValue>ERROR_COMMAND_ABORTED</allowedValue>
      <allowedValue>ERROR_NOT_ENABLED_FOR_INTERNET</allowedValue>
      <allowedValue>ERROR_USER_DISCONNECT</allowedValue>
      <allowedValue>ERROR_ISP_DISCONNECT</allowedValue>
      <allowedValue>ERROR_IDLE_DISCONNECT</allowedValue>
      <allowedValue>ERROR_FORCED_DISCONNECT</allowedValue>
      <allowedValue>ERROR_NO_CARRIER</allowedValue>
      <allowedValue>ERROR_IP_CONFIGURATION</allowedValue>
      <allowedValue>ERROR_UNKNOWN</allowedValue>
   </allowedValueList>
</stateVariable>

<stateVariable sendEvents="no">
   <name>AutoDisconnectTime</name>
   <dataType>ui4</dataType>
</stateVariable>

<stateVariable sendEvents="no">
   <name>IdleDisconnectTime</name>
   <dataType>ui4</dataType>
</stateVariable>

<stateVariable sendEvents="no">
   <name>WarnDisconnectDelay</name>
   <dataType>ui4</dataType>
</stateVariable>

<stateVariable sendEvents="no">
   <name>RSIPAvailable</name>
   <dataType>boolean</dataType>
</stateVariable>

<stateVariable sendEvents="no">
   <name>NATEnabled</name>
   <dataType>boolean</dataType>
</stateVariable>

<stateVariable sendEvents="yes">
   <name>ExternalIPAddress</name>
   <dataType>string</dataType>
</stateVariable>

<stateVariable sendEvents="yes">
   <name>PortMappingNumberOfEntries</name>
   <dataType>ui2</dataType>
</stateVariable>

<stateVariable sendEvents="no">
   <name>PortMappingEnabled</name>
   <dataType>boolean</dataType>
</stateVariable>

<stateVariable sendEvents="no">
   <name>PortMappingLeaseDuration</name>
   <dataType>ui4</dataType>
   <defaultValue>Vendor-defined</defaultValue>
   <allowedValueRange>
      <minimum>0</minimum>
      <maximum>604800</maximum>
   </allowedValueRange>
</stateVariable>

<stateVariable sendEvents="no">
   <name>RemoteHost</name>
```

```
         <dataType>string</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>ExternalPort</name>
         <dataType>ui2</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>InternalPort</name>
         <dataType>ui2</dataType>
         <allowedValueRange>
            <minimum>1</minimum>
            <maximum>65535</maximum>
         </allowedValueRange>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>PortMappingProtocol</name>
         <dataType>string</dataType>
         <allowedValueList>
            <allowedValue>TCP</allowedValue>
            <allowedValue>UDP</allowedValue>
         </allowedValueList>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>InternalClient</name>
         <dataType>string</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>PortMappingDescription</name>
         <dataType>string</dataType>
      </stateVariable>

      <stateVariable sendEvents="yes">
         <name>SystemUpdateID</name>
         <dataType>ui4</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Manage</name>
         <dataType>boolean</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_PortListing</name>
         <dataType>string</dataType>
      </stateVariable>

      <!-- Declarations for other state variables defined by UPnP vendor (if any)
go here -->

   </serviceStateTable>
</scpd>
```


Annex A
(informative)

Theory of Operation

WANIPConnection service is used to control IP connections and related functions. Its operation can be divided into three categories:

- Connection management,
- Status information of the gateway,
- NAT traversal control.

The connection management starts with initializing a *WANDevice*, which is initialized with one or more instances of *WANConnectionDevice* depending on the number of physical links the gateway is configured to support. For example, cable modem would typically implement one *WANConnectionDevice* instance, but multiple instances may exist for supporting VCs in the case of DSL.

Refer to the *WANPPPConnection* service definition for more details on connection setup procedures. A table summarizing connection procedures follows.

Table A.1 — Connection Procedures

Value of *ConnectionType*	Control point capabilities	Step N°	Follow-up steps for a control point
IP_Routed	IP Stack	1	Set the default gateway address to the Internet Gateway address
		2	Send IP packets through the gateway
IP_Bridged	IP Stack	1	Get the ISP IP address (through DHCP?) and set it as the default gateway address.
		2	Send IP packets to ISP IP address

A.1.1 Connection Scenarios

As previously mentioned, the possible connection types for a *WANIPConnection* are *IP_Routed* and *IP_Bridged*. The connection scenarios for these two types of connections and the role of connection related actions are described in more detail below.

A.1.1.1 *IP_ROUTED*

Unlike the *WANPPPConnection*, a *WANIPConnection* instance typically does not require a priori configuration. If the *IP_Routed* connection is the default connection on the IGD, a control point on the LAN that desires to use the connection is not required to send the *RequestConnection()* action even if the connection is not *Connected*. If the connection is *Disconnected*, the IGD will initiate a WAN connection upon receiving any outbound packets from the control point (assuming the 'dial-on-demand' option is enabled on the IGD) or upon receiving a *RequestConnection()* action. This may cause the IGD to obtain an IP address via DHCP from the ISP. It results in a transition of *ConnectionStatus* to *Connected*. The IGD shares the routable WAN IP address with control points on the LAN using Network Address Translation (NAT). The control points on the LAN are assigned private IP addresses in response to their DHCP requests (control points may self-assign non-routable IP addresses in certain IGD configurations).

If the IGD supports multiple WAN connection instances, the *RequestConnection()* action is intended for a control point to specify a *WANIPConnection* instance (that in all likelihood is different from the default connection).

A control point may use *RequestTermination()* or *ForceTermination()* to disconnect the IGD from the WAN (this involves releasing any previously acquired IP resources from the ISP).

RequestTermination(): A control point can invoke this action, if available, to terminate an active connection. As an example, if three control points were sharing a WAN connection

instance and if each were to call *RequestTermination()*, the IGD may release IP resources acquired from the ISP on the three instances of *RequestTermination()* to conserve IP resources. If *WarnDisconnectDelay* is implemented and is non-zero the IGD is required to change the *ConnectionStatus* from *Connected* to *PendingDisconnect* and wait until *WarnDisconnectDelay* seconds elapse before transitioning to the *Disconnected* state.

ForceTermination(): The IGD will immediately release all WAN IP resources, disregarding the value of *WarnDisconnectDelay* variable.

An example of an implementation of this connection type is a routing IGD modeling a PC or embedded gateway with a cable modem as a WAN interface.

A.1.1.2 *IP_BRIDGED*

In this scenario, all Ethernet packets from a control point on the LAN are bridged to the WAN by the IGD. If this were the default connection, all Ethernet traffic across all LAN interfaces will be bridged to the WAN side. The actions *RequestConnection()*, *RequestTermination()* and *ForceTermination()* are not relevant in this case since the IGD is not IP addressable by the control point over the LAN. Therefore, UPnP IGD would be inactive and not in use.

If the gateway is IP addressable, it is possible to use UPnP IGD for gateway configuration with appropriate actions. For instance, a control point may use the *RequestConnection()* action to select a specific WAN connection instance, followed typically by a DHCP renewal request. All Ethernet packets (including DHCP requests) from this control point get redirected (bridged) through the default WAN connection. This assumes that the IGD is capable of source (MAC) address based bridging. The control point that is actively using the connection may issue *RequestTermination()* or *ForceTermination()* actions through a secondary interface (if the control point is multi-homed) to end the use of this connection and change the *ConnectionStatus* to *Disconnected*. Alternatively, a control point that is not using the connection may issue *RequestTermination()* or *ForceTermination()* to disconnect IGD from the WAN.

An example of an implementation of this scenario would be a bridging IGD with an integrated cable modem on the WAN interface that, in turn, has an Ethernet link to CM Termination System (CMTS). If an IGD supports multiple WAN connection instances and has one active (IP) bridged connection, it cannot allow other WAN connections to be simultaneously active unless it supports source (MAC) address based bridging on that bridged connection, where the source MAC address identifies a control point. The *RequestConnection()* action returns an error if this were the case.

A.1.2 Non-UPnP compliant clients

The gateway should support non-UPnP compliant devices by making it possible for a client to start accessing the Internet (effectively Dial-on-Demand) without sending *RequestConnection()* command. The client in this scenario cannot specify which particular *WANConnectionDevice* or *WANIPConnection* it wants to use. The *WANIPConnection* to be used is identified using the *DefaultConnectionService* identified in *Layer3Forwarding* service. Also, the client will not be able to terminate the connection or use the other features of *WANIPConnection* service (like detecting connection speed or specifying a new port mapping).

A.1.3 VPN connections

VPN sessions may be established on an IP connection initiated at the gateway. There are two cases to consider:

- A VPN client is initiated by a client on the residential LAN. In this case, the VPN is transparent to the *WANIPConnection* instance and is not visible in the UPnP context.
- A VPN client is initiated on the gateway. In this case, the VPN session would use an *WANIPConnection* instance. A VPN service to model this scenario is not standardized in this Working Committee – it is possible however, as a vendor extension. One possible way to do this is to provide a VPN service in *InternetGatewayDevice* outside of *WANDevice*. The state table for this service would support configuration attributes that are essential for setting up a VPN connection. Thesewould include parameters such as:

- IP address(es) of VPN Gateway,
- Security Protocols to be used,
- Authentication and Privacy parameters specific to a security protocol,
- Session time-out delay.

In addition, it would also contain a *ConnectionService* variable that specifies a *WANIPConnection* service instance in a *WANConnectionDevice*. A comma-separated 2-tuple uniquely identifies the service:

uuid:**device-UUID**:*WANConnectionDevice*:v , urn:**upnp-org**:**serviceId**:*serviceID.*

The VPN service would support a *RequestConnection()* action that would in turn invoke the *RequestConnection()* of the corresponding *WANIPConnection* service like any other UPnP client.

NOTES:

- For *IP_Bridged* connections, it is assumed that either all LAN ports (*LANDevices*) or none of the LAN ports are bridged to the connection. *RequestConnection()* is a NOP in this case.
- In the case of Always-On IP connections, an implementation may return an appropriate error code if *ForceTermination()* is not supported.

A.2 NAT & NAT traversal

A.2.1 NAT & NAPT

NAT (Network Address Translation) is a popular tool for alleviating the IPv4 address shortage. For example, NAT allows to enable multiple hosts on a private network to access the Internet using a single public IP address.

NAT involves re-writing the source and/or destination IP addresses, and usually also the TCP/UDP port numbers of IP packets as they pass through the NAT. Checksums (both IP and TCP/UDP) shall also be rewritten to take into account the changes. A NAT which also rewrites the ports is called a NAPT (Network Address Port Translation). NAPT is the most widely used NAT mechanism in residential gateways.

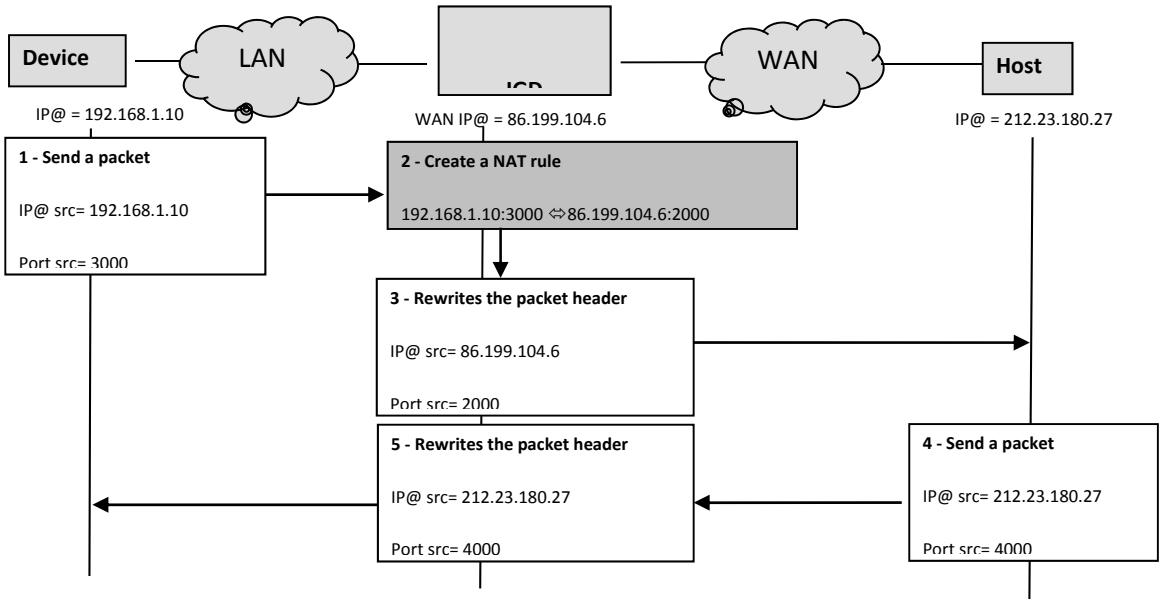

Figure A.1 — NAT is an IP address translator

A.2.2 NAT traversal

Even if this solution is largely used in the residential gateways which access the WAN, it does not solve all the problems. There are some remaining issues like:

- **Expiration of the mapping timer:** In order to manage its resources, a NAT destroys a NAT rule after a period of inactivity between the internal endpoint and the external endpoint,
- **IP addresses carried in payloads:** Protocols like SIP, H.323, SNMP do not respect the layered model, and they re-use the IP packet's address in their payload. NAT causes problems for these protocols, because the private IP addresses in the payload are no longer valid once a packet crosses the NAT,
- **Issues with bundled session applications:** Bundled session applications are applications which use a control connection to establish a media stream. such as FTP (in passive mode), H.323, SIP and RTSP, for example. In the example below, the device contacts a host on the WAN, and this remote host is responsible for establishing the session on another port. It does not work because there is no NAT binding for the requested session.

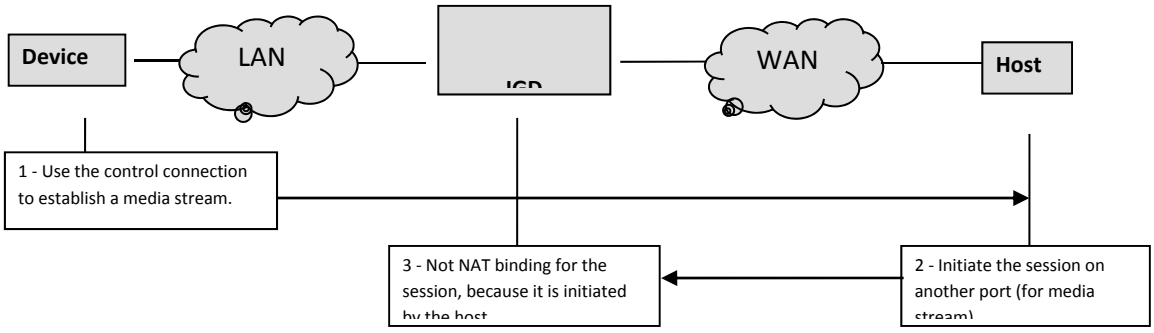

Figure A.2 — NAT issue with bundled session applications

UPnP IGD provides a compelling NAT traversal mechanism to cope with those NAT issues by allowing an application (e.g. a control point) to control the mapping timer, to create NAT rules for upcoming session on another port and to get the external IP address of the residential gateway in order to use it directly in the payload.

A.3 UPnP IGD & NAT Traversal

The purpose of NAT traversal and port mappings is to support the programmatic creation of short-lived dynamic port mappings from any control point on the residential network for applications such as multiplayer games, Internet chat, and Peer-to-Peer messaging that use external ports for short session-based communication.

A port mapping is essentially an 8-tuple of the type:

<*PortMappingEnabled*, *PortMappingLeaseDuration*, *RemoteHost*, *ExternalPort*, *InternalPort*, *PortMappingProtocol*, *InternalClient*, *PortMappingDescription*>

The port mapping is used by clients to enable forwarding of inbound service requests, if NAT is used as the address translation mechanism between the residential (private) LAN and the Internet. Each 8-tuple configures NAT to listen for packets on the external interface of the *WANConnectionDevice* on behalf of a specific client and dynamically forward connection requests to that client.

If a firewall is co-resident on the gateway, it is assumed that the gateway will appropriately configure the firewall for the port mapping.

For example, a client on a residential LAN could run an HTTP server and configure the gateway to forward requests from specific hosts on the Internet (WAN) on specific WAN interfaces.

Annex B
(informative)

Bibliography

The following documents, in whole or in part, may be useful for understanding this document but they are not essential for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[13] IETF RFC 3102, Realm Specific IP: Framework, M.Borella, J. Lo, D. Grabelsky, G. Montenegro, October 2001.
Available at: http://tools.ietf.org/html/rfc3102.

[14] IETF RFC 3103, Realm Specific IP: Protocol Specification, M. Borella, D. Grabelsky, J. Lo, K. Taniguchi, October 2001.
Available at: http://tools.ietf.org/html/rfc3103.

[15] IETF RFC 5382, NAT Behavioral Requirements for TCP, S. Guha, K. and Biswas and B. Ford and S. Sivakumar and P. Srisuresh, October 2008.
Available at: http://tools.ietf.org/html/rfc5382.

[16] IETF RFC 4787, Network Address Translation (NAT) Behavioral Requirements for Unicast UDP, F. Audet and C. Jennings, January 2007.
Available at: http://tools.ietf.org/html/rfc4787.

ICS 35.200

Price based on 65 pages