WANIPConnection:1 Service Template Version 1.01

For UPnP™ Version 1.0
Status: Standardized DCP
Date: November 12, 2001

Authors	**Company**
Matthew Schmitz	Cisco Systems, Inc.
Ulhas Warrier, Prakash Iyer	Intel Corporation

Contents

1. OVERVIEW AND SCOPE 4

1.1. CHANGE LOG 4

2. SERVICE MODELING DEFINITIONS 7

2.1. SERVICETYPE 7
2.2. STATE VARIABLES 7
2.2.1. ConnectionType....................................*10*
2.2.2. PossibleConnectionTypes *10*
2.2.3. ConnectionStatus....................................*10*
2.2.4. Uptime....................................*10*
2.2.5. LastConnectionError....................................*10*
2.2.6. AutoDisconnectTime *10*
2.2.7. IdleDisconnectTime *11*
2.2.8. WarnDisconnectDelay....................................*11*
2.2.9. RSIPAvailable....................................*11*
2.2.10. NATEnabled *11*
2.2.11. ExternalIPAddress....................................*11*
2.2.12. PortMappingNumberOfEntries *12*
2.2.13. PortMappingEnabled *12*
2.2.14. PortMappingLeaseDuration *12*
2.2.15. RemoteHost *12*
2.2.16. ExternalPort *12*
2.2.17. InternalPort *12*
2.2.18. PortMappingProtocol....................................*12*
2.2.19. InternalClient....................................*12*
2.2.20. PortMappingDescription....................................*13*
2.2.21. Relationships Between State Variables *14*
2.3. EVENTING AND MODERATION....................................14
2.3.1. Event Model....................................*15*
2.4. ACTIONS....................................16
2.4.1. SetConnectionType....................................*16*
2.4.2. GetConnectionTypeInfo *17*
2.4.3. RequestConnection....................................*17*
2.4.4. RequestTermination *18*
2.4.5. ForceTermination....................................*19*
2.4.6. SetAutoDisconnectTime *20*
2.4.7. SetIdleDisconnectTime *20*
2.4.8. SetWarnDisconnectDelay....................................*21*
2.4.9. GetStatusInfo *21*
2.4.10. GetAutoDisconnectTime *22*
2.4.11. GetIdleDisconnectTime *22*
2.4.12. GetWarnDisconnectDelay....................................*23*
2.4.13. GetNATRSIPStatus *23*
2.4.14. GetGenericPortMappingEntry *24*
2.4.15. GetSpecificPortMappingEntry....................................*25*
2.4.16. AddPortMapping....................................*25*
2.4.17. DeletePortMapping....................................*27*
2.4.18. GetExternalIPAddress *27*
2.4.19. Non-Standard Actions Implemented by a UPnP Vendor....................................*28*
2.4.20. Relationships Between Actions....................................*28*

2.4.21. Common Error Codes........28
2.5. THEORY OF OPERATION........28
2.5.1. Connection Initiation........29
2.5.2. Connection Termination........30
2.5.3. Connection Scenarios........32
2.5.4. Non-UPnP compliant clients........33
2.5.5. VPN connections........33

3. XML SERVICE DESCRIPTION........34

4. TEST........42

List of Tables

Table 1: State Variables........7

Table *1.1:* allowedValueList for `PossibleConnectionTypes`........8

Table *1.2:* allowedValueList for `ConnectionStatus`........8

Table *1.3:* allowedValueList for `LastConnectionError`........9

Table *1.4:* allowedValueList for `PortMappingProtocol`........10

Table 2: Event Moderation........14

Table 3: Actions........16

Table 4: Arguments for `SetConnectionType`........16

Table 5: Arguments for `GetConnectionTypeInfo`........17

Table 6: Arguments for `SetAutoDisconnectTime`........20

Table 7: Arguments for `SetIdleDisconnectTime`........20

Table 8: Arguments for `SetWarnDisconnectDelay`........21

Table 9: Arguments for `GetStatusInfo`........21

Table 10: Arguments for `GetAutoDisconnectTime`........22

Table 11: Arguments for `GetIdleDisconnectTime`........22

Table 12: Arguments for `GetWarnDisconnectDelay`........23

Table 13: Arguments for `GetNATRSIPStatus`........23

Table 14: Arguments for `GetGenericPortMappingEntry`........24

Table 15: Arguments for `GetSpecificPortMappingEntry`........25

Table 16: Arguments for `AddPortMapping`........26

Table 17: Arguments for `DeletePortMapping`........27

Table 18: Arguments for `GetExternalIPAddress`........27

Table 19: Common Error Codes...28

1. Overview and Scope

This service definition is compliant with the UPnP Device Architecture version *1.0*

This service-type enables a UPnP control point to configure and control IP connections on the WAN interface of a UPnP compliant ***InternetGatewayDevice***[*]. Any type of WAN interface (e.g., DSL or Cable) that can support a IP connection can use this service.

The service is REQUIRED if an IP connection is used for WAN access, and is specified in
urn:schemas-upnp-org:device:*WANConnectionDevice*
one or more instances of which are specified under the device
urn:schemas-upnp-org:device:*WANDevice*

An instance of ***WANDevice*** is specified under the root device
urn:schemas-upnp-org:device:*InternetGatewayDevice*

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 SST below) 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 UPnP Architecture version 1.0, 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.

1.1. Change Log

WANConnection:0.5 was replaced by connection services specific to the type of WAN physical access. This SCP specifically covers Cable-based connections.
Changes from *WANConnection:0.5*

- Renamed the service to *WANCableConnection:0.6* as per decision in WC to maintain separate connection service for each access type.
- Removed POTS related references and discussion on Dial-up connections.
- Updated return codes for error.

Changes from *WANCableConnection:0.6*

[*] Refer to companion documents defined by the UPnP Internet Gateway working committee for more details on specific devices and services referenced in this document.

- Added 'Get' actions per Technical Committee recommendation to not use QueryStateVariable for reading state variables.

Changes from *WANCableConnection:0.7*

- Removed reference to Service Instance Identifier.
- Renamed IPConnectionStatus to ConnectionStatus
- Removed securityEstablished, ipnotReady values for the ConnectionStatus and added dsSyncComplete, usParamAcquired, rangingComplete substates.
- Added RestartCableInterface action set
- Simplified Theory of Operation section
- Added a Reference section
- Added ConfigFile
- Removed TODServer

Changes from *WANCableConnection:0.8*

- Removed Cable modem specific variables/actions and descriptions, renamed to *WANIPConnection*
- Changed connection state values, added AutoDisconnectTime and IdleDisconnectTime and WarnDisconnectDelay
- Added variables to reflect NAT settings and service mapping variables

Changes from *WANIPConnection:0.9*

- Added text to clarify usage models in Theory of Operation section (based on paper from Frédéric Pennerath)
- Added variables ConnectionType and PossibleConnectionTypes to SST. Also added corresponding actions.

Changes from *WANIPConnection:0.91*

- Changed address range delimited for service map address range from '/' to '\'.

Changes from *WANIPConnection:0.92*

- Removed white spaces from XML section.

Changes from *WANIPConnection:0.93*

- Text clarifications in various sections.
- Changed formal parameters to actions to differentiate between them and related state variables
- Updated document status to template design complete

Changes from *WANIPConnection:0.94*

- Updated to service template v1.01
- Verified against TDC checklist v1.01
- Changed WarnDisconnectDelay, AutoDisconnectTime and IdleDisconnectTime from Required to Optional
- Changed section on NAT port mappings by adding new SST variables and actions as discussed in the WC
- GetNatInfo was replaced by GetNATRSIPStatus
- Replaced GetLinkLayerInfo with 2 Get actions: GetLinkLayerMaxBitRates and GetLinkLayerDetails
- Updated XML section to reflect these changes
- Split actions that deal with optional variables to handle individual variables separately

Changes from *WANIPConnection:0.8*

- Added </action> tag before GetConnectionTypeInfo in XML template
- Removed default values for SST variables and updated XML template accordingly
- Deleted Vendor Defined rows from allowedValueList tables
- Required versus Optional changes in allowedValueList tables
- Deleted error code 712 from actions – this is a catastrophic error that is not possible in a correct IGD implementation
- Added error code 714 to GetSpecificPortMappingEntry
- Updated text for error codes 710 and 718
- Replaced text referring to service mapping with port mapping
- Defined wildcard for IP addresses as an empty string and for ports as 0

- Added new required action – GetExternalIPAddress
- Added text to clarify conflict issues with NAT port mappings
- Deleted white spaces from XML section

Changes from *WANIPConnection:0.81*

- Added XML comment tags to comments text in XML template
- Deleted white space for ERROR_NOT_ENABLED_FOR_INTERNET tag in XML template
- Added text for Unconfigured value of ConnectionStatus
- Added clarifying text to RequestConnection action
- Added clarifying text related to PortMappingLeaseDuration
- Minor clarifications on the use of wildcards for port mapping tuple variables
- Added error code 501 to AddPortMapping and fixed error description for 501
- Minor updates to state diagram for IP connections

Changes from *WANIPConnection:0.82*

- Updated semantic tests section
- Removed allowedvaluerange specification in XML template for Uptime, AutoDisconnectTime, IdleDisconnectTime, WarnDisconnectDelay
- Added clarification text for allowedvaluelist table for PossibleConnectionTypes
- Deleted A_PortMappingIndex variable and replaced its occurrences with NewPortMappingIndex
- Text clarifications in Theory of operation section for Connection Initiation

Changes from *WANIPConnection:0.9*

- Added allowed value list to PortMappingProtocol in XML section
- Include PendingDisconnect and Disconnecting also as other starting states for ForceTermination action in section 2.4.5. Updated figure 1 to include more starting states for ForceTermination action.
- Added new section “Connection Scenarios” to clarify behavior of connection related actions
- Changed support of non-UPnP client (section 2.5.5) to a ‘should’ (from ‘must’).
- Added error codes 726 and 727 to AddPortMapping.
- Removed restricted applicability to IP_Routed connection type from text describing RequestConnection

Changes from *WANIPConnection:0.99*

- Changed allowed value range for InternalPort to “1 to 65535 inclusive”

Changes from *WANIPConnection:0.991*

- Copyright messages and document status updated.

2. Service Modeling Definitions

2.1. ServiceType

The following service type identifies a service that is compliant with this template:

urn:schemas-upnp-org:service:*WANIPConnection:1*.

2.2. State Variables

Table 1: State Variables

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value [2]	Default Value [2]	Eng. Units
`ConnectionType`	R	string	Depends on `PossibleConnectionTypes`	Not specified	N/A
`PossibleConnectionTypes`	R	string	See Table 1.1	Not specified	N/A
`ConnectionStatus`	R	string	See Table 1.2	Not specified	N/A
`Uptime`	R	ui4	Undefined	Not specified	seconds
`LastConnectionError`	R	string	See Table 1.3	Not specified	N/A
`AutoDisconnectTime`	O	ui4	>= 0	Not specified	seconds
`IdleDisconnectTime`	O	ui4	>= 0	Not specified	seconds
`WarnDisconnectDelay`	O	ui4	>= 0	Not specified	seconds
`RSIPAvailable`	R	boolean	0, 1	Not specified	N/A
`NATEnabled`	R	boolean	0,1	Not specified	N/A
`ExternalIPAddress`	R	string	String of the type “x.x.x.x”	Empty string	N/A
`PortMappingNumberOfEntries`	R	ui2	>=0	Not specified	N/A
`PortMappingEnabled`	R	boolean	0,1	Not specified	N/A
`PortMappingLeaseDuration`	R	ui4	0 to maximum value of ui4	Not specified	seconds
`RemoteHost`	R	string	String of the type “x.x.x.x” or empty string	Empty string	N/A
`ExternalPort`	R	ui2	Between 0 and 65535 inclusive	Not specified	N/A
`InternalPort`	R	ui2	Between 1 and 65535 inclusive	Not specified	N/A
`PortMappingProtocol`	R	string	See Table 1.4	Empty string	N/A

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value [2]	Default Value [2]	Eng. Units
`InternalClient`	R	string	String of the type “x.x.x.x”	Empty string	N/A
`PortMappingDescription`	R	string	Undefined	Empty string	N/A
Non-standard state variables implemented by an UPnP vendor go here.	*X*	*TBD*	*TBD*	*TBD*	*TBD*

[1] R = Required, O = Optional, X = Non-standard.

[2] Values listed in this column are required. To specify standard optional values or to delegate assignment of values to the vendor, you must reference a specific instance of an appropriate table below.

NOTE: Default values are not specified in the DCP. A vendor may however choose to provide default values for SST variables where appropriate.

Table 1.1: allowedValueList for `PossibleConnectionTypes`

PLEASE NOTE: PossibleConnectionTypes is defined as a comma-separated string. However, the values within the string are restricted to the list given in the table below. We have used the allowedValueList table format only as a convenience to represent these values.

Value	Req. or Opt. [3]	Description
Unconfigured	R	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 DEFERED TO THE VENDOR.**
IP_Routed	R	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. SHOULD NOT BE TESTED IN OTHER DEVICE CONFIGURATIONS.**
IP_Bridged	R	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. SHOULD NOT BE TESTED IN OTHER DEVICE CONFIGURATIONS.**

[3]

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

Table 1.2: allowedValueList for `ConnectionStatus`

Value	Req. or Opt. [3]	Description
Unconfigured	R	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`.
Connecting	O	The ***WANConnectionDevice*** is in the process of initiating a connection for the first time after the connection became disconnected.
Connected	R	At least one client has successfully initiated an Internet connection using this instance.
PendingDisconnect	O	The connection is active (packets are allowed to flow through), but will transition to *Disconnecting* state after a certain period (indicated by `WarnDisconnectDelay`).
Disconnecting	O	The ***WANConnectionDevice*** is in the process of terminating a connection. On successful termination, `ConnectionStatus` transitions to *Disconnected*.
Disconnected	R	No ISP connection is active (or being activated) from this connection instance. No packets are transiting the gateway.

[3]

NOTE: Whether or not a control point gets notified of the intermediary states of a connection transition may depend on the gateway implementation.

Table 1.3: allowedValueList for `LastConnectionError`

Value	Req. or Opt. [3]
ERROR_NONE	R
ERROR_COMMAND_ABORTED	O
ERROR_ NOT_ENABLED_FOR_INTERNET	O
ERROR_USER_DISCONNECT	O
ERROR_ISP_DISCONNECT	O
ERROR_IDLE_DISCONNECT	O
ERROR_FORCED_DISCONNECT	O
ERROR_NO_CARRIER	O

ERROR_IP_CONFIGURATION	O
ERROR_UNKNOWN	O

[3]

Table 1.4: allowedValueList for `PortMappingProtocol`

Value	Req. or Opt. [3]
TCP	R
UDP	R

[3]

2.2.1. `ConnectionType`

This variable is set to specify the connection type for a specific active connection. The value selected must be one from the list specified in `PossibleConnectionTypes`.

2.2.2. `PossibleConnectionTypes`

This variable represents a comma-separated string 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 1.1

2.2.3. `ConnectionStatus`

This variable represents current status of an Internet connection. Possible string values are specified in table 1.2

2.2.4. `Uptime`

This variable represents the time in seconds that this connection has stayed up.

2.2.5. `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 1.3

2.2.6. `AutoDisconnectTime`

This 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 device policy
- `EnabledForInternet` variable (see ***WANCommonInterfaceConfig***[*]) being set to 0 by a user 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***.

2.2.7. `IdleDisconnectTime`

It 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.
Note: Layer 2 heartbeat packets are included as part of an idle state i.e., they do not reset the idle timer
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***.

2.2.8. `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.

2.2.9. `RSIPAvailable`

This variable indicates if Realm-specific IP (RSIP) is available as a feature on the ***InternetGatewayDevice***. RSIP is being defined in the NAT working group in the IETF 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 should set this variable to 0.

2.2.10. `NATEnabled`

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

2.2.11. `ExternalIPAddress`

This is the external IP address used by NAT for the connection.

[*] Refer to companion document defined by the UPnP Internet Gateway working committee for more details on this variable

2.2.12.`PortMappingNumberOfEntries`

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

2.2.13.`PortMappingEnabled`

This variable allows security conscious users to disable and enable dynamic and static NAT port mappings on the IGD.

2.2.14.`PortMappingLeaseDuration`

This variable determines the time to live in seconds of a port-mapping lease. A value of 0 means the port mapping is static. Non-zero values will allow support for dynamic port mappings. Note that static port mappings do not necessarily mean persistence of these mappings across device resets or reboots. It is up to a gateway vendor to implement persistence as appropriate for their IGD device.

2.2.15.`RemoteHost`

This variable represents the source of inbound IP packets. This will be a wildcard in most cases (i.e. an empty string). NAT vendors are only required to support 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`. When `RemoteHost` is specified as one 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.

2.2.16.`ExternalPort`

This variable 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.

2.2.17.`InternalPort`

This variable 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.

2.2.18.`PortMappingProtocol`

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

2.2.19.`InternalClient`

This variable represents the IP address or DNS host name of an internal client (on the residential LAN). 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 mandatory and support for DNS host names is recommended. This value cannot be a wildcard (i.e. empty string). It must 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.

2.2.20.`PortMappingDescription`

This is a string representation of a port mapping and is applicable for static and dynamic port mappings. 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) may or may not be unique across multiple instantiations of an application on multiple nodes in the residential LAN.

The purpose of NAT port mappings is 2-fold:

- To support the programmatic creation of static port mappings from any control point on the residential network to enable a majority of network services and applications that listen on well known ports.
- 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.

These mappings are represented as an array of entries.

Following details about NAT port mappings are worth noting:

Adding / Creating a New Port Mapping:

If the mapping contains a unique `ExternalPort` and `PortMappingProtocol` pair the addition will be successful, unless the NAT is out of resources.

Overwriting Previous / Existing Port Mappings:

If the `RemoteHost`, `ExternalPort`, `PortMappingProtocol` and `InternalClient` are exactly the same as an existing mapping, the existing mapping values for `InternalPort`, `PortMappingDescription`, `PortMappingEnabled` and `PortMappingLeaseDuration` are overwritten.

Rejecting a New Port Mapping:

In cases where the `RemoteHost`, `ExternalPort` and `PortMappingProtocol` are the same as an existing mapping, but the `InternalClient` is different, the `AddPortMapping` action is rejected with an appropriate error.

Add or Reject New Port Mapping behavior based on vendor implementation:
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.

2.2.21.Relationships Between 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.

For dynamic port mappings (i.e. port mappings with a finite lease duration), the `PortMappingLeaseDuration` variable counts down from the value set by the `AddPortMapping` 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 IGD, **independent** of the state of `PortMappingEnabled` for that specific port mapping. The IGD 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 decrement by 1, which will be evented. Dynamic port mappings will not be automatically reinitiated by the IGD – it is the responsibility of a control point to reinstall 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.

`PortMappingLeaseDuration` does not change for static port mappings (i.e. mappings with infinite lease duration) **independent** of the state of `PortMappingEnabled` variable.

2.3. Eventing and Moderation

Table 2: Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
`ConnectionType`	No	No	N/A	N/A	N/A
`PossibleConnectionTypes`	Yes	No	N/A	N/A	N/A
`ConnectionStatus`	Yes	No	N/A	N/A	N/A
`Uptime`	No	No	N/A	N/A	N/A

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
`LastConnectionError`	No	No	N/A	N/A	N/A
`AutoDisconnectTime`	No	No	N/A	N/A	N/A
`IdleDisconnectTime`	No	No	N/A	N/A	N/A
`WarnDisconnectDelay`	No	No	N/A	N/A	N/A
`RSIPAvailable`	No	No	N/A	N/A	N/A
`NATEnabled`	No	No	N/A	N/A	N/A
`ExternalIPAddress`	Yes	No	N/A	N/A	N/A
`PortMappingNumberOfEntries`	Yes	No	N/A	N/A	N/A
`PortMappingEnabled`	No	No	N/A	N/A	N/A
`PortMappingLeaseDuration`	No	No	N/A	N/A	N/A
`RemoteHost`	No	No	N/A	N/A	N/A
`ExternalPort`	No	No	N/A	N/A	N/A
`InternalPort`	No	No	N/A	N/A	N/A
`PortMappingProtocol`	No	No	N/A	N/A	N/A
`InternalClient`	No	No	N/A	N/A	N/A
`PortMappingDescription`	No	No	N/A	N/A	N/A
Non-standard state variables implemented by an UPnP vendor go here.	*TBD*	*TBD*	*TBD*	*TBD*	*TBD*

[1] Determined by N, where Rate = (Event)/(N secs).
[2] (N) * (allowedValueRange Step).

2.3.1. Event Model

Eventing is self-explanatory. Clients use event updates on `ConnectionStatus` to provide local user feedback and manage connections initiated by local applications. None of the events are moderated.

2.4. Actions

Immediately following this table is detailed information about these actions, including short descriptions of the actions, the effects of the actions on state variables, and error codes defined by the actions.

Table 3: Actions

Name	Req. or Opt. [1]
`SetConnectionType`	R
`GetConnectionTypeInfo`	R
`RequestConnection`	R
`RequestTermination`	O
`ForceTermination`	R
`SetAutoDisconnectTime`	O
`SetIdleDisconnectTime`	O
`SetWarnDisconnectDelay`	O
`GetStatusInfo`	R
`GetAutoDisconnectTime`	O
`GetIdleDisconnectTime`	O
`GetWarnDisconnectDelay`	O
`GetNATRSIPStatus`	R
`GetGenericPortMappingEntry`	R
`GetSpecificPortMappingEntry`	R
`AddPortMapping`	R
`DeletePortMapping`	R
`GetExternalIPAddress`	R
Non-standard actions implemented by an UPnP vendor go here.	X

[1] R = Required, O = Optional, X = Non-standard.

2.4.1. `SetConnectionType`

This action sets up a specific connection type. Clients on the LAN may initiate or share connection only after this action completes or `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.

2.4.1.1. Arguments

Table 4: Arguments for `SetConnectionType`

Argument	Direction	relatedStateVariable
`NewConnectionType`	IN	`ConnectionType`

2.4.1.2. Dependency on State (if any)

2.4.1.3. Effect on State (if any)

This action sets the connection to a specific type.

2.4.1.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
703	InactiveConnection StateRequired	Current value of `ConnectionStatus` should be either *Disconnected* or *Unconfigured* to permit this action.

2.4.2. `GetConnectionTypeInfo`

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

2.4.2.1. Arguments

Table 5: Arguments for `GetConnectionTypeInfo`

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

2.4.2.2. Dependency on State (if any)

2.4.2.3. Effect on State (if any)

None.

2.4.2.4. Errors

ErrorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.

2.4.3. `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 may be obvious, it is worth noting that a control point must 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 following implementation guidelines are also worth noting:

- The IGD should implement a timeout mechanism to ensure that it does not remain in the `Connecting` state forever. The timeout values are 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.
- Control points should manage a timeout for initiated connections to recover from catastrophic failures on the IGD. The timeout values are implementation dependent.

See the 'Theory of Operation' section below for more details.

2.4.3.1. Arguments

This action does not have any arguments.

2.4.3.2. Dependency on State (if any)

2.4.3.3. Effect on State (if any)

If successful, `ConnectionStatus` is changed to Connected.

2.4.3.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
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.
710	InvalidConnectionType	This action is not permitted for the specified `ConnectionType`.

2.4.4. `RequestTermination`

A client may send this command to any connection instance in *Connected* or *Connecting* state to change `ConnectionStatus` to *Disconnected*. Connection state changes to *PendingDisconnect* depending on the value of `WarnDisconnectDelay` variable. Connection termination will depend on whether other clients intend to continue to use the connection. The process of terminating a connection is described in Theory of Operation section.

2.4.4.1. Arguments

This action does not have any arguments.

2.4.4.2. Dependency on State (if any)

2.4.4.3. Effect on State (if any)

If successful, `ConnectionStatus` is changed to Disconnected.

2.4.4.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
707	DisconnectInProgress	The connection is in the process of being torn down.
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.

2.4.5. `ForceTermination`

A client may send this command to any connection instance in *Connected*,*Connecting, PendingDisconnect or Disconnecting* state to change `ConnectionStatus` to *Disconnected*. Connection state immediately transitions to *Disconnected* irrespective of the setting of `WarnDisconnectDelay` variable. The process of terminating a connection is described in Theory of Operation section.

2.4.5.1. Arguments

This action does not have any arguments.

2.4.5.2. Dependency on State (if any)

2.4.5.3. Effect on State (if any)

If successful, `ConnectionStatus` is changed to Disconnected.

2.4.5.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
707	DisconnectInProgress	The connection is in the process of being torn down.
710	InvalidConnectionType	This command is valid only when `ConnectionType` is *IP-Routed*

711	ConnectionAlready Terminated	An attempt was made to terminate a connection that is no longer active.

2.4.6. `SetAutoDisconnectTime`

This action sets the time (in seconds) after which an active connection is automatically disconnected.

2.4.6.1. Arguments

Table 6: Arguments for `SetAutoDisconnectTime`

Argument	Direction	relatedStateVariable
`NewAutoDisconnectTime`	*IN*	`AutoDisconnectTime`

2.4.6.2. Dependency on State (if any)

2.4.6.3. Effect on State (if any)

After expiration of specified time, `ConnectionStatus` is changed to Disconnected.

2.4.6.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.

2.4.7. `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.

2.4.7.1. Arguments

Table 7: Arguments for `SetIdleDisconnectTime`

Argument	Direction	relatedStateVariable
`NewIdleDisconnectTime`	*IN*	`IdleDisconnectTime`

2.4.7.2. Dependency on State (if any)

2.4.7.3. Effect on State (if any)

After the time specified in seconds expires, connection termination is initiated. The intermediate connection states before the connection is terminated will depend on `WarnDisconnectDelay.`

2.4.7.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.

2.4.8. `SetWarnDisconnectDelay`

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

2.4.8.1. Arguments

Table 8: Arguments for `SetWarnDisconnectDelay`

Argument	Direction	relatedStateVariable
`NewWarnDisconnectDelay`	IN	`WarnDisconnectDelay`

2.4.8.2. Dependency on State (if any)

2.4.8.3. Effect on State (if any)

After the time specified in seconds expires, the connection is terminated.

2.4.8.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.

2.4.9. `GetStatusInfo`

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

2.4.9.1. Arguments

Table 9: Arguments for `GetStatusInfo`

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

2.4.9.2. Dependency on State (if any)

2.4.9.3. Effect on State (if any)

None.

2.4.9.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.

2.4.10.`GetAutoDisconnectTime`

This action retrieves the values of various timeouts related to the termination of a connection.

2.4.10.1.Arguments

Table 10: Arguments for `GetAutoDisconnectTime`

Argument	Direction	relatedStateVariable
`NewAutoDisconnectTime`	*OUT*	`AutoDisconnectTime`

2.4.10.2.Dependency on State (if any)

2.4.10.3.Effect on State (if any)

None.

2.4.10.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.

2.4.11.`GetIdleDisconnectTime`

This action retrieves the values of various timeouts related to the termination of a connection.

2.4.11.1.Arguments

Table 11: Arguments for `GetIdleDisconnectTime`

Argument	Direction	relatedStateVariable
`NewIdleDisconnectTime`	*OUT*	`IdleDisconnectTime`

2.4.11.2.Dependency on State (if any)

2.4.11.3.Effect on State (if any)

None.

2.4.11.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.

2.4.12.`GetWarnDisconnectDelay`

This action retrieves the values of various timeouts related to the termination of a connection.

2.4.12.1.Arguments

Table 12: Arguments for `GetWarnDisconnectDelay`

Argument	Direction	relatedStateVariable
`NewWarnDisconnectDelay`	*OUT*	`WarnDisconnectDelay`

2.4.12.2.Dependency on State (if any)

2.4.12.3.Effect on State (if any)

None.

2.4.12.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.

2.4.13.`GetNATRSIPStatus`

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

2.4.13.1.Arguments

Table 13: Arguments for `GetNATRSIPStatus`

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

2.4.13.2.Dependency on State (if any)

2.4.13.3.Effect on State (if any)

None.

2.4.13.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.

2.4.14.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.`

2.4.14.1.Arguments

Table 14: 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`

2.4.14.2.Dependency on State (if any)

2.4.14.3.Effect on State (if any)

None.

2.4.14.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.

713	SpecifiedArrayIndexInvalid	The specified array index is out of bounds

2.4.15.GetSpecificPortMappingEntry

This action reports the Static Port Mapping specified by the unique tuple of `RemoteHost`, `ExternalPort` and `PortMappingProtocol`.

2.4.15.1.Arguments

Table 15: 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

2.4.15.2.Dependency on State (if any)

2.4.15.3.Effect on State (if any)

None.

2.4.15.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
714	NoSuchEntryInArray	The specified value does not exist in the array

2.4.16.AddPortMapping

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.

NOTE: Not all NAT implementations will support:

- Wildcard value (i.e. 0) for `ExternalPort`
- `InternalPort` values that are different from `ExternalPort`
- Dynamic port mappings i.e. with non-Infinite `PortMappingLeaseDuration`

2.4.16.1.Arguments

Table 16: Arguments for `AddPortMapping`

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

2.4.16.2.Dependency on State (if any)

2.4.16.3.Effect on State (if any)

None.

2.4.16.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
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	Internal and External port values must be the same

725	OnlyPermanentLeasesSupported	The NAT implementation only supports permanent lease times on port mappings
726	RemoteHostOnlySupportsWildcard	RemoteHost must be a wildcard and cannot be a specific IP address or DNS name
727	ExternalPortOnlySupportsWildcard	ExternalPort must be a wildcard and cannot be a specific port value

2.4.17.`DeletePortMapping`

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

2.4.17.1.Arguments

Table 17: Arguments for `DeletePortMapping`

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

2.4.17.2.Dependency on State (if any)

2.4.17.3.Effect on State (if any)

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

2.4.17.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
714	NoSuchEntryInArray	The specified value does not exist in the array

2.4.18.`GetExternalIPAddress`

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

2.4.18.1.Arguments

Table 18: Arguments for `GetExternalIPAddress`

Argument	Direction	relatedStateVariable
`NewExternalIPAddress`	*OUT*	`ExternalIPAddress`

2.4.18.2.Dependency on State (if any)

2.4.18.3.Effect on State (if any)

None.

2.4.18.4.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.

2.4.19.Non-Standard Actions Implemented by a UPnP Vendor

To facilitate certification, non-standard actions implemented by UPnP vendors should be included in this service template. The UPnP Device Architecture lists naming requirements for non-standard actions (see the section on Description).

2.4.20.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.

2.4.21.Common Error Codes

The following table 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 19: Common Error Codes

errorCode	errorDescription	Description
401	Invalid Action	See UPnP Device Architecture section on Control.
402	Invalid Args	See UPnP Device Architecture section on Control.
404	Invalid Var	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
600-699	TBD	Common action errors. Defined by UPnP Forum Technical Committee.
701-799		Common action errors defined by the UPnP Forum working committees.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.5. Theory of Operation

When a ***WANDevice*** is initialized, it is intialized 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.

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

2.5.1. 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 be 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*.

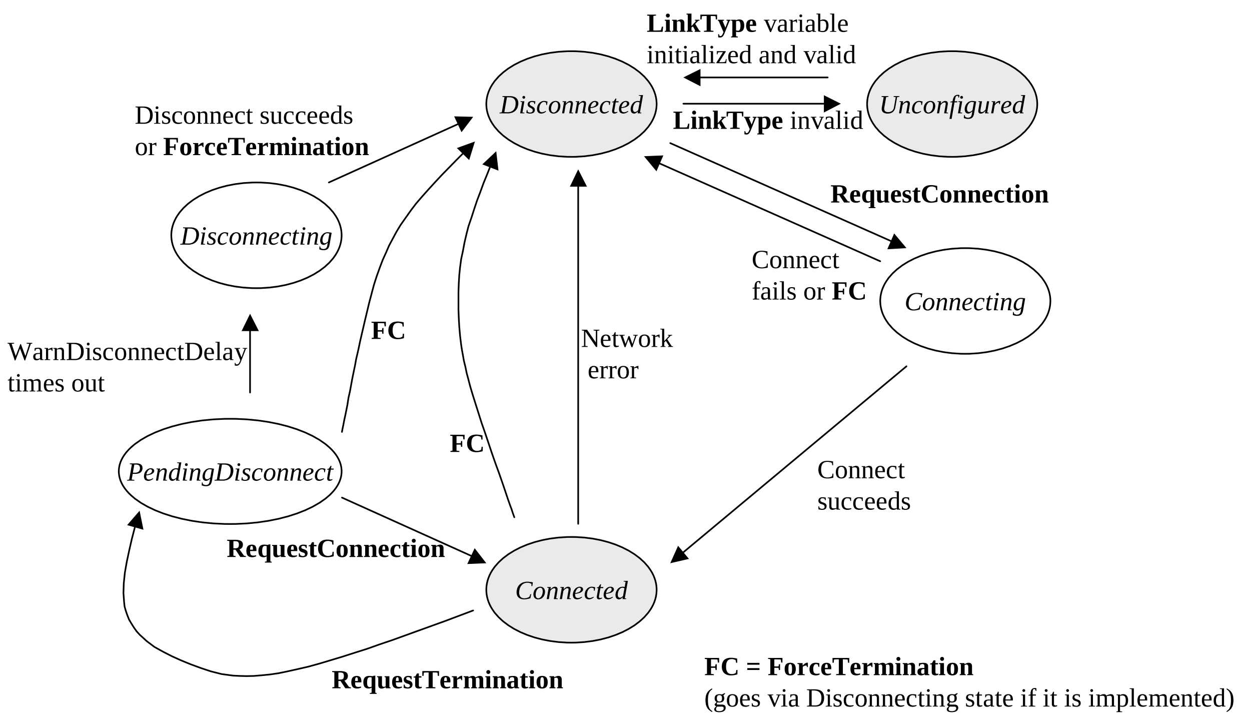

Figure 1 State Diagram for IP Connections

When a connection service gets a `RequestConnection` command, if the `ConnectionStatus` is:

- *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.*
- *Connected*: the client is allowed to use the connection if `ConnectionType` is *IP_routed,* otherwise an error is returned.

Figure 1 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:

1. Network failure
2. `ConnectionStatus` is *Connecting*
3. `EnabledForInternet` variable in ***WANCommonInterfaceConfig*** is set to 0 (false)

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

2.5.2. 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 connection termination command is acted upon only if the `ConnectionType` is *IP_routed* and `ConnectionStatus` is *Connecting or Connected.*

A connection termination may be initiated due to:

1. A `RequestTermination` or `ForceTermination` command from a client
2. `AutoDisconnectTime` or `IdleDisconnectTime` coming into effect
3. A deployment specific Gateway policy
4. `EnabledForInternet` variable (in ***WANCommonInterfaceConfig*** service) being set to 0
5. 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 and `RequestTermination` is called): This occurs if `WarnDisconnectDelay` is non-zero and the cause for termination is 1 or 2 (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 (3 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
    - 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 optional 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 *Disconnected* directly – it does not go to *PendingDisconnect* even if `WarnDisconnectDelay` is non-zero.

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.

2.5.3. 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.

2.5.3.1. 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 CP on the LAN that desires to use the connection is not required to send the `RequestConnection` action even if the connection is not *active*. If the connection is *inactive*, the IGD will initiate a WAN connection upon receiving any outbound packets from the CP (assuming the 'dial-on-demand' option is enabled on the IGD) **or** upon receiving a `RequestConnection` action. This may translate translate to the IGD obtaining an IP address via DHCP from the ISP. It results in a transition of `ConnectionStatus` to *active*. The IGD shares the routable WAN IP address with CPs on the LAN using Network Address Translation (NAT). The CPs on the LAN are assigned private IP addresses in response to their DHCP requests (CPs 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 CP to specify a ***WANIPConnection*** instance (that in all likelihood is different from the default connection).
A CP may use `RequestTermination` or `ForceTermination` to disconnect the IGD from the WAN (this involves releasing any previously acquired IP resources from the ISP).

<u>RequestTermination</u>: A CP can invoke this action, if available, to terminate an *active* connection. As an example, if three CPs 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.

<u>ForceTermination</u>: 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.

2.5.3.2. IP_Bridged

In this scenario, all Ethernet packets from a CP 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 CP over the LAN.
If this were not the default, a CP 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 CP get redirected (bridged) through the default WAN connection. This assumes that that the IGD is capable of source (MAC) address based bridging. The CP that is actively using the connection may issue `RequestTermination` or `ForceTermination` actions through a secondary interface (if the CP is multi-homed) to end the use of this connection and change the `ConnectionStatus` to *inactive*. Alternatively, a CP 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 CP. The `RequestConnection` action returns an error if this were the case.

2.5.4. 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).

2.5.5. VPN connections

VPN sessions may be established on an IP connection initiated at the gateway. There are 2 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 WC – 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. These would 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 (***LANDevice***s) 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.

3. 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>
    <!-- Declarations for other actions added by UPnP vendor (if any) go
here -->
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="no">
      <name>ConnectionType</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="yes">
      <name>PossibleConnectionTypes</name>
      <dataType>string</dataType>
      <allowedValueList>
        <allowedValue>Unconfigured</allowedValue>
        <allowedValue>IP_Routed</allowedValue>
        <allowedValue>IP_Bridged</allowedValue>
      </allowedValueList>
    </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>
    </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>
    </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>
    <!-- Declarations for other state variables added by UPnP vendor (if
any) go here -->
  </serviceStateTable>
</scpd>
```

4. Test

SetConnectionType / GetConnectionTypeInfo

Test Sequence 1: To test success path
Semantic class: 4
Pre-condition:

- Connection must be inactive. To verify, call `GetStatusInfo` and check OUT argument `ConnectionStatus`. Value should be Unconfigured or Disconnected.

GetConnectionTypeInfo Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
ConnectionType	NA			
PossibleConnectionTypes	Initialized to a list of allowable connection types (see Table 1.1)			
		Error Code (if any)	NA	NA

SetConnectionType Success=200

In-Arg	Values	State Variables	Current State	Expected State
ConnectionType	Must be one of the values returned in PossibleConnectionTypes	ConnectionStatus*	Unconfigured	Disconnected
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

* The state change on `ConnectionStatus` will not occur if the current state is already set to Disconnected.

Test Sequence 2: To test Set followed by Get
Semantic class: 1
Pre-condition: None
Same as test sequence 1, followed by the following:

GetConnectionTypeInfo Success=200

In-Arg	Values	State Variables	Current State	Expected

				State
		NA	NA	NA
Out-Arg	Expected Value			
ConnectionType	Set in previous SetConnectionType action			
PossibleConnectionTypes	Initialized to a list of allowable connection types (see Table 1.1)			
		Error Code (if any)	NA	NA

Test Sequence 3: To test error 703
Semantic class: 4
Pre-conditions:

- If `ConnectionStatus` is set to Unconfigured, dependent variables such as `ConnectionType` may have to be initialized first
- If `EnabledForInternet` is implemented and set to 0, action `SetEnabledForInternet` in WANCommonInterfaceConfig MUST be invoked first to set the value to 1 prior to invoking `RequestConnection`.
- WAN connectivity must be provisioned to allow `RequestConnection` to complete successfully.
- For DSL-integrated IGD Only: If the device does NOT support AutoConfig, `LinkType` in `WANDSLLinkConfig` MUST be set to a valid value PRIOR to executing the above sequence of actions

GetStatusInfo Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
ConnectionStatus	Not Unconfigured			
LastConnectionError	NA			
Uptime	NA			
		Error Code (if any)	NA	NA

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected

				State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

SetConnectionType Success=200

In-Arg	Values	State Variables	Current State	Expected State
ConnectionType	Must be one of the values returned in PossibleConnectionTypes	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	703	NA

RequestConnection

Test Sequence 4: To test success path
Semantic class: 3
Pre-conditions:

- IGD settings (e.g. `LinkType`) should be pre-configured and WAN connectivity provisioned as described earlier, to enable `RequestConnection` to succeed.
- If `EnabledForInternet` is implemented in `WANCommonInterfaceConfig`, it should be set to 1 prior to executing this sequence of actions.

GetStatusInfo Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
ConnectionStatus	Disconnected			
LastConnectionError	NA			
Uptime	NA			
		Error Code (if any)	NA	NA

4.1.1.1.

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

GetStatusInfo **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
ConnectionStatus	Connected			
LastConnectionError	ERROR_NONE			
Uptime	NA			
		Error Code (if any)	NA	NA

Test Sequence 5: To test error 704
Semantic class: 3
Pre-conditions:

- The IGD must be physically disconnected from the ISP/headend or the WAN link must be in use prior to running the following test sequence
- IGD settings (e.g. `LinkType`) should be pre-configured to otherwise enable `RequestConnection` to succeed

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	No change
Out-Arg	Expected Value			
		Error Code (if any)	704	NA

GetStatusInfo **Success=200**

In-Arg	Values	State Variables	Current State	Expected State

		NA	NA	NA
Out-Arg	Expected Value			
ConnectionStatus	Connected			
LastConnectionError	Valid error code; see below			
Uptime	NA			
		Error Code (if any)	NA	NA

Some examples of possible error values for `LastConnectionError` are ERROR_NO_DIALTONE or ERROR_LINE_BUSY

Test Sequence 6: To test error 706
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Unconfigured.

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	No change
Out-Arg	Expected Value			
		Error Code (if any)	706	NA

Test Sequence 7: To test error 705
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.
- IGD settings (e.g. `LinkType`) should be pre-configured and WAN connectivity provisioned as described earlier, to enable `RequestConnection` to succeed.

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	706	NA

RequestConnection **Success=200** Executed in sequence with no time delay

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	705	NA

NOTE: It may not be possible to reproduce this test in certain deployments where connection setup is almost instantaneous.

Test Sequence 8: To test error 707
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.
- IGD settings (e.g. `LinkType`) should be pre-configured and WAN connectivity provisioned as described earlier, to enable `RequestConnection` to succeed.

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

ForceTermination **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

`ConnectionStatus` will change to Disconnecting and eventually Disconnected and will be evented.

RequestConnection **Success=200** Executed in sequence with no time delay

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			

		Error Code (if any)	707	NA

NOTE: It may not be possible to reproduce this test in certain deployments where connection teardown is almost instantaneous.

Test Sequence 9: To test error 709
Semantic class: 3
Pre-conditions:

- Vendor must implement `SetEnabledForInternet` and related actions in the WANCommonInterfaceConfig service.

SetEnabledForInternet Success=200 in WANCommonInterfaceConfig

In-Arg	Values	State Variables	Current State	Expected State
EnabledForInternet	0	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	709	NA

Test Sequence 10: To test error 708
Semantic class: 3
Pre-conditions:

- POTS IGD Only: `SetISPInfo` in POTSLinkConfig with empty `ISPPhoneNumber`. The action should succeed.
- DSL-integrated IGD Only: `SetDestinationAddress` in WANDSLLinkConfig to invalid value.

SetISPInfo Success=200 POTS IGD Only in WANPOTSLinkConfig

In-Arg	Values	State Variables	Current State	Expected State
ISPPhoneNumber	Empty string	NA	NA	NA
ISPInfo	NA			
LinkType	NA			

Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

SetDestinationAddress Success=200 DSL IGD Only in WANDSLLinkConfig

In-Arg	Values	State Variables	Current State	Expected State
DestinationAddress	Empty string	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	708	NA

Test Sequence 11: To test error 710
Semantic class: 3
Pre-conditions: None

SetConnectionType Success=200

In-Arg	Values	State Variables	Current State	Expected State
ConnectionType	Must be one of the values returned in PossibleConnectionTypes but incompatible with RequestConnection. An example is PPPoE_Bridged	NA	NA	NA
Out-Arg	Expected Value	ConnectionStatus	Disconnected	No change
		Error Code (if any)	NA	NA

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	710	NA

RequestConnection / SetAutoDisconnectTime

Test Sequence 12: To test success path
Semantic class: 3
Pre-conditions:
Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

SetAutoDisconnectTime Success=200

In-Arg	Values	State Variables	Current State	Expected State
AutoDisconnectTime	30	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

After 30 seconds, `ConnectionStatus` will change to Disconnecting and eventually Disconnected and will be evented.

RequestConnection / SetIdleDisconnectTime

Test Sequence 13: To test success path
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

SetIdleDisconnectTime Success=200

In-Arg	Values	State Variables	Current State	Expected State
IdleDisconnectTime	30	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

After 30 seconds of no IP traffic on the connection, `ConnectionStatus` will change to Disconnecting and eventually Disconnected and will be evented.

NOTE: `IdleDisconnectTime` requires no traffic for specified period of time in seconds, which may be difficult to reproduce.

RequestConnection /SetWarnDisconnectDelay

Test Sequence 14: To test success path
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

SetAutoDisconnectTime **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
AutoDisconnectTime	30	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

SetWarnDisconnectDelay **Success=200**

In-Arg	Values	State Variables	Current State	Expected State

WarnDisconnectDelay	30	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

After 30 seconds, `ConnectionStatus` will change to PendingDisconnect and eventually will be evented. After 15 seconds, `ConnectionStatus` will change to Disconnected and will be evented.

RequestTermination

Test Sequence 15: To test success path
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestTermination Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

NOTE: This test sequence presumes that the connection is configured a priori. If not, follow steps to configure the connection

Test Sequence 16: To test error 711
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestTermination **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestTermination **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	711	NA

Test Sequence 17: To test error 707
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected

				State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestTermination **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestTermination **Success=200** Executed in sequence with no time delay

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	707	NA

NOTE: This test may not be possible in certain deployments where connection teardown is almost instantaneous.

Test Sequence 18: To test error 710
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

SetConnectionType **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
ConnectionType	Must be one of the values returned in PossibleConnectionTypes but incompatible with RequestConnection. An example is PPPoE_Bridged	NA	NA	NA

Out-Arg	Expected Value	ConnectionStatus	Disconnected	No change
		Error Code (if any)	NA	NA

Follow steps to activate the connection (i.e. `ConnectionStatus` is Connected).

RequestTermination Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	710	NA

RequestTermination / SetWarnDisconnectDelay

Test Sequence 19: To test success path
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

SetWarnDisconnectDelay Success=200

In-Arg	Values	State Variables	Current State	Expected State
WarnDisconnectDelay	30	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestTermination Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Pending Disconnect (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

After 30 seconds, `ConnectionStatus` will change to Disconnected and will be evented.

ForceTermination

Test Sequence 20: To test success path
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

ForceTermination **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

NOTE: This test sequence presumes that the connection is configured a priori. If not, follow steps to configure the connection

Test Sequence 21: To test error 711
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

ForceTermination Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

ForceTermination Success=200

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	711	NA

Test Sequence 22: To test error 707
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

RequestConnection Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

ForceTermination Success=200

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

ForceTermination **Success=200** Executed in sequence with no time delay

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	707	NA

NOTE: This test may not be possible in certain deployments where connection teardown is almost instantaneous.

Test Sequence 23: To test error 710
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

SetConnectionType **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
ConnectionType	Must be one of the values returned in PossibleConnectionTypes but incompatible with RequestConnection. An example is PPPoE_Bridged	NA	NA	NA
Out-Arg	Expected Value	ConnectionStatus	Disconnected	No change
		Error Code (if any)	NA	NA

Follow steps to activate the connection (i.e. `ConnectionStatus` is Connected).

ForceTermination **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA

Out-Arg	Expected Value			
		Error Code (if any)	710	NA

ForceTermination / SetWarnDisconnectDelay

Test Sequence 24: To test the fact that WarnDisconnectDelay has no effect on ForceTermination
Semantic class: 3
Pre-conditions:

- Follow sequence of actions outlined earlier to ensure that `ConnectionStatus` is Disconnected.

SetWarnDisconnectDelay **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
WarnDisconnectDelay	30	NA	NA	NA
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

RequestConnection **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Disconnected	Connected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

ForceTermination **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		ConnectionStatus	Connected	Disconnected (evented)
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

AddPortMapping / DeletePortMapping

Test Sequence 25: To test success path
Semantic class: 2

Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

GetPortMappingNumberOfEntries **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			
PortMappingNumberOfEntries	0 or a positive integer	Error Code (if any)	NA	NA

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

GetPortMappingNumberOfEntries **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
		NA	NA	NA
Out-Arg	Expected Value			

PortMappingNumberOfEntries	1 more than the value retrieved prior to the AddPortMapping action	Error Code (if any)	NA	NA

Test Sequence 26: To test error 718
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	No change
ExternalPort	80			

PortMappingProtocol	TCP			
InternalPort	81			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	718	NA

Test Sequence 27: To test success path with DeletePortMapping
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

DeletePortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	A positive integer	Decrement by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

Action should cause PortMappingNumberOfEntries to decrement by 1 and will be evented.

Test Sequence 28: To test error 714
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

AddPortMapping Success=200

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

DeletePortMapping Success=200

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	A positive integer	Decrement by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

DeletePortMapping Success=200

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	No change
ExternalPort	80			
PortMappingProtocol	TCP			
Out-Arg	Expected Value			
		Error Code (if any)	714	NA

Test Sequence 29: To test error 724
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

NOTE: This test is ONLY for implementations that do not support different values for `ExternalPort` and `InternalPort`.

AddPortMapping Success=200

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	No change
ExternalPort	85			

PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	724	NA

Test Sequence 30: To test error 725
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

NOTE: This test is ONLY for implementations that do not support dynamic port mappings (i.e. those with finite lease durations).

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	No change
ExternalPort	85			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	2000			
Out-Arg	Expected Value			
		Error Code (if any)	725	NA

AddPortMapping / GetGenericPortMapping / GetSpecificPortMapping

Test Sequence 31: To test success path
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	81			
PortMappingProtocol	TCP			
InternalPort	81			

InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	81			
PortMappingProtocol	TCP			
InternalPort	81			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

GetGenericPortMappingEntry Success = 200

In-Arg	Values	State Variables	Current State	Expected State
A_PortMappingIndex	0 to 2			
Out-Arg	Expected Value			
RemoteHost	Values should correspond to			

	those previously added			
ExternalPort	Values should correspond to those previously added			
PortMappingProtocol	Values should correspond to those previously added			
InternalPort	Values should correspond to those previously added			
InternalClient	Values should correspond to those previously added			
PortMappingEnabled	Values should correspond to those previously added			
PortMappingDescription	Values should correspond to those previously added			
PortMappingLeaseDuration	Values should correspond to those previously added			
		Error Code (if any)	NA	NA

GetSpecificPortMappingEntry Success = 200

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	Values should correspond to those previously added			
ExternalPort	Values should correspond to			

	those previously added			
PortMappingProtocol	Values should correspond to those previously added			
Out-Arg	Expected Value			
InternalPort	Values should correspond to those previously added			
InternalClient	Values should correspond to those previously added			
PortMappingEnabled	Values should correspond to those previously added			
PortMappingDescription	Values should correspond to those previously added			
PortMappingLeaseDuration	Values should correspond to those previously added			
		Error Code (if any)	NA	NA

Test Sequence 32: To test error 713
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive	Increment by 1 (evented)

			integer	
ExternalPort	80			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	81			
PortMappingProtocol	TCP			
InternalPort	81			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

GetGenericPortMappingEntry Success = 200

In-Arg	Values	State Variables	Current State	Expected

				State
A_PortMappingIndex	2			
Out-Arg	**Expected Value**			
RemoteHost	NA			
ExternalPort	NA			
PortMappingProtocol	NA			
InternalPort	NA			
InternalClient	NA			
PortMappingEnabled	NA			
PortMappingDescription	NA			
PortMappingLeaseDuration	NA			
		Error Code (if any)	713	NA

Test Sequence 33: To test error 714
Semantic class: 2
Pre-conditions:

- Port mapping entry being added should not already exist in the port mapping table. Values provided below serve only as an example.

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	80			
PortMappingProtocol	TCP			
InternalPort	80			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			

PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

AddPortMapping **Success=200**

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	A valid IP address	PortMappingNumberOfEntries	0 or a positive integer	Increment by 1 (evented)
ExternalPort	81			
PortMappingProtocol	TCP			
InternalPort	81			
InternalClient	A valid IP address			
PortMappingEnabled	1			
PortMappingDescription	Test Description			
PortMappingLeaseDuration	0			
Out-Arg	Expected Value			
		Error Code (if any)	NA	NA

GetSpecificPortMappingEntry Success = 200

In-Arg	Values	State Variables	Current State	Expected State
RemoteHost	Values should correspond to those previously added			
ExternalPort	Values should correspond to those previously added			
PortMappingProtocol	5000			

Out-Arg	Expected Value			
InternalPort	Values should correspond to those previously added			
InternalClient	Values should correspond to those previously added			
PortMappingEnabled	Values should correspond to those previously added			
PortMappingDescription	Values should correspond to those previously added			
PortMappingLeaseDuration	Values should correspond to those previously added			
		Error Code (if any)	714	NA

Change History

Change Log for Version 1.0 (10-4-00)

- Revised the Title Page to call out V1.0 of the Service Template
- Changed to be consistent with Sample Designs released to the Technical Committee
- Service State Table: Variable Descriptions removed from the table and are listed in specific sections following the table.
- Actions: Reformatted the information contained in the Action Table:
    - Added overview entry point.
    - Added an Action Summary Table to specify Required or Optional
    - Added enumerated sections to specify each actions: Arguments, Effect on State, and Errors.