RemoteUIClient:1 Service Template Version 1.01

For UPnP™ Version *1.0*
Status: *Standardized DCP*
Date: *September 2, 2004*

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

Authors	Company
Ylian Saint-Hilaire	Intel Corporation
Mark R. Walker	Intel Corporation
Jan van Ee	Philips

Contents

List of Tables

1. Overview and Scope

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

This service-type encapsulates the management of an out-of-band remoting protocol connection to a device capable of user interaction. This service is required for all Remote UI clients.

It is specified in: **urn:schemas-upnp-org:device:*RemoteUIClientDevice***

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:*RemoteUIClient:1*.

2.2. State Variables

Table 1: Service State Variables

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value	Default Value [2]	Eng. Units
DeviceProfile	R	string	Undefined	Empty string	N/A
CurrentConnections	R	string	Undefined	Empty string	N/A
CurrentConnectionsEvent	O	string	Undefined	Empty string	N/A
CompatibleUIsUpdateIDEvent	O	i4	Undefined	Undefined	N/A
A_ARG_TYPE_String	O	string	Undefined	Empty string	N/A
A_ARG_TYPE_CompatibleUIs	O	string	Undefined	Empty string	N/A
A_ARG_TYPE_DisplayMessageType	O	string	Undefined	Empty string	N/A
A_ARG_TYPE_InputDataType	O	string	Undefined	Empty string	N/A
A_ARG_TYPE_Int	O	i4	Undefined	Undefined	N/A
Non-standard state variables implemented by a UPnP vendor go here.	*X*	*TBD*	*TBD*	*TBD*	*TBD*

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

[2] Default 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.

2.2.1. CompatibleUIsUpdateIDEvent

CompatibleUIsUpdateIDEvent is an optional, evented integer state variable used to indicate a change in the state of the *A_ARG_TYPE_CompatibleUIs* listing optionally stored on the client device. *CompatibleUIsUpdateIDEvent* is incremented and evented every time a change in *A_ARG_TYPE_CompatibleUIs* occurs. The value rolls over to one once it reaches the maximum value. The maximum value is *2147483647*.

2.2.2. CurrentConnections

This required variable is a comma-separated list representing all current Remote UI client sessions. The values of *CurentConnections* include the *ConnectionsUpdateID* quantity in the first field. An example of a *CurrentConnections* value is contained in section 3.1.2.

2.2.3. CurrentConnectionsEvent

This optional state variable is the evented form of the required state variable *CurrentConnections*.

2.2.4. DeviceProfile

DeviceProfile values are UTF-8 XML-formatted strings used by the client device to represent the list of all supported remoting protocols. An example *DeviceProfile* value appears in section 3.1.4. The XML schema defining the format of *DeviceProfile* values appears in section 4.

2.2.5. A_ARG_TYPE_CompatibleUIs

An *A_ARG_TYPE_CompatibleUI*s value is a string formatted as UTF-8 XML representing the list of all UIs that are compatible with a designated Remote UI client device. Remote UI client devices must be able to support values of *A_ARG_TYPE_CompatibleUIs* that are 10k bytes in length. Client support for longer values is optional. An example *A_ARG_TYPE_CompatibleUIs* value appears in section 3.1.3. The XML schema defining the format of *A_ARG_TYPE_CompatibleUIs* values in contained in section 5.

2.2.6. A_ARG_TYPE_DisplayMessageType

An *A_ARG_TYPE_DisplayMessageType* value is a string formatted as a MIME type of a message to be displayed using the optional **DisplayMessage()** action. If this action is implemented, the value *text/plain* must be included in the list of allowed values for this state variable. Allowed values for *A_ARG_TYPE_DisplayMessageType* are listed in table 2.

Table 2: Allowed value list for state variable `A_ARG_TYPE_DisplayMessageType`

Value	Description
text/plain	Indicates client device support for messages of *text/plain* type.
Vendor_Specified_MIME_Type	*Client device vendors are permitted to enumerate additional supported message MIME types.*

2.2.7. A_ARG_TYPE_InputDataType

A string specifying the type of the input used in the optional **ProcessInput()** action. Client device vendors implementing one or more of the standard input types listed in table 3 are encouraged to use the allowed values for this state variable, also listed in table 3.

Table 3: Allowed value list for state variable `A_ARG_TYPE_InputDataType`

Value	Description
ASCII	Indicates that inputs of this type consist of two hexadecimal digits, corresponding to individual ASCII characters.
UNICODE	Indicates that inputs of this type consist of 4 hexadecimal digits, corresponding to individual unicode values.
ISO10646	Indicates that inputs of this type consist of 8 hexadecimal digits,

	corresponding to values defined in ISO 10646.
ISO8859-1	Indicates that inputs of this type consist of two hexadecimal digits, corresponding to values defined in ISO8859-1.
Vendor_CodeName	*Client device vendors are permitted to enumerate additional input data types.*

2.2.8. A_ARG_TYPE_Int

A simple 4 byte integer.

2.2.9. A_ARG_TYPE_String

A simple string type.

2.3. Eventing and Moderation

Table 4: Event moderation

Variable Name	Evented	Moderated Event	Max Event Rate [1]	Logical Relation	Min Delta per Event [2]
CurrentConnections	No	N/A	N/A	N/A	N/A
CurrentConnectionsEvent	Yes	No	N/A	N/A	N/A
DeviceProfile	No	N/A	N/A	N/A	N/A
A_ARG_TYPE_String	No	N/A	N/A	N/A	N/A
A_ARG_TYPE_InputDataType	No	N/A	N/A	N/A	N/A
A_ARG_TYPE_DisplayMessageType	No	N/A	N/A	N/A	N/A
A_ARG_TYPE_CompatibleUIs	No	N/A	N/A	N/A	N/A
CompatibleUIsUpdateIDEvent	Yes	No	N/A	N/A	N/A
A_ARG_TYPE_Int	No	N/A	N/A	N/A	N/A

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

2.3.1. Relationships Between State Variables

CurrentConnections is a required state variable and ***CurrentConnectionsEvent*** is an optional state variable that is used to event ***CurrentConnections*** when its value changes. Client device vendors may chose to issue an event when ***CurrentConnections*** changes by supporting ***CurrentConnectionsEvent***. Control points receiving the event will effectively be automatically notified of a change in the connection status of the issuing client device.

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.

Table 5: Actions

Name	Req. or Opt. [1,2]
Connect	R
Disconnect	R
GetCurrentConnections	R
GetDeviceProfile	R
GetUIListing	O
AddUIListing	O[2]
RemoveUIListing	O[2]
DisplayMessage	O
ProcessInput	O
Non-standard actions implemented by an UPnP vendor go here.	X

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

[2] When implemented, the optional actions: **AddUIListing()** and **RemoveUIListing()** must be implemented together and must also include the **GetUIListing()** action. **GetUIListing()** however, may be implemented without implementing **AddUIListing()** and **RemoveUIListing()**.

2.4.1. Connect

This required action results in the establishment of a new connected session on the client device. It also optionally modifies the sessions on hold for client devices that support placing sessions on hold. **Connect()** waits for and confirms that the requested out-of-band connection listed in the input argument has been successfully established.

2.4.1.1. Arguments

Table 6: Arguments for Connect( )

Argument	Direction	relatedStateVariable
RequestedConnections	IN	CurrentConnections
CurrentConnectionsList	OUT	CurrentConnections

- Only one new connection is allowed in the ***RequestedConnections*** input argument. **Connect()** fails (with error code 701) if more than one new connection appears in the ***RequestedConnections*** input argument.
- **Connect()** fails (with error code 705) if the value of *ConnectionsUpdateID* in the first field of ***RequestedConnections*** does not match the local value stored on the Remote UI client.
- The fields following the required *ConnectionsUpdateID* value in the ***RequestedConnections*** input argument are optional. These optional fields correspond to the requested configuration of the on-hold UIs. The ordering of the on-hold UI values indicates the requested hold-stack ordering after the new session is established.

- ***CurrentConnectionsList*** lists the new *ConnectionsUpdateID* value followed by the currently active session, and for clients that support on-hold sessions, the configuration of the on-hold sessions resulting from the successful **Connect()** action.
- **Connect()** is always successful if the new connection indicated in ***RequestedConnections*** is successfully established. Inability to comply with the requested ordering of the on-hold UI sessions does not cause **Connect()** to fail.
- **Connect()** fails with error code 707 if the Remote UI client determines that the requested new connection is invalid or non-routable.

2.4.1.2. Dependency on State (if any)

2.4.1.3. Effect on State

2.4.1.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
701	More Than One New Session	**Connect()** failed because more than one new session requested for connection is contained in the input argument.
702	Already Connected	**Connect()** failed because the remote UI client is already connected to this user interface.
703	UI Server Failure	**Connect()** failed because the out-of-band remoting protocol failed to establish a session due to a UI server failure.
704	Session Connection Timeout	**Connect()** failed because the out-of-band remoting protocol failed to establish a session in the time allotted..
705	*ConnectionUpdateID* Value Mismatch	**Connect()** failed because the value of *ConnectionsUpdateID* contained in the first field of the input argument does not match the current value of *ConnectionsUpdateID* stored on the Remote UI client device.
706	Max Hold Capacity Exceeded	**Connect()** failed because it would have caused the number of on-hold sessions to exceed the capacity of the Remote UI client.
707	Operation Rejected	**Connect()** failed because the Remote UI Client has rejected the operation.

2.4.2. Disconnect

This required action disconnects one or more sessions. The sessions specified for disconnection can either be active or on hold.

2.4.2.1. Arguments

Table 7: Arguments for `Disconnect()`

Argument	Direction	relatedStateVariable
RequestedDisconnects	*IN*	CurrentConnections
CurrentConnectionsList	*OUT*	CurrentConnections

- All Remote UI sessions specified in the ***RequestedDisconnects*** listing following the *ConnectionsUpdateID* value must correspond to active sessions or sessions on hold. **Disconnect()** fails (with error code 711) if one or more UI sessions appearing in the ***RequestedDisconnects*** input argument is not an active session or a session on hold.

- If the active connection is disconnected or placed on hold as a result of the **Disconnect()** action, the session that was at the top of the on-hold list prior to the action becomes the newly active session by default, if the client supports on-hold sessions.

- ***CurrentConnectionsList*** lists the new *ConnectionsUpdateID* value followed by the currently active session. For clients that support on-hold UI sessions, the optional fields of the ***CurrentConnectionsList*** argument following the active session indicate the configuration of the on-hold sessions resulting from the successful **Disconnect()** action.

- **Disconnect()** fails (with error code 705) if the value of *ConnectionsUpdateID* in the first field of ***RequestedDisconnects*** does not match the local value stored on the Remote UI client.

2.4.2.2. Dependency on State (if any)

The success or failure of this action depends on the type of operation that is performed and the currently active user interface connection.

2.4.2.3. Effect on State

2.4.2.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
711	Unknown UI	**Disconnect()** failed because one or more of the UIs specified in the ***RequestedDisconnects*** input argument did not correspond to an active or on-hold UI.
705	*ConnectionUpdateID* Value Mismatch	**Disconnect()** failed because the value of *ConnectionsUpdateID* contained in the first field of the input argument does not match the current value of *ConnectionsUpdateID* stored on the Remote UI client device.

2.4.3. GetCurrentConnections

This action retrieves the ***CurrentConnections*** listing from the Remote UI client.

2.4.3.1. Arguments

Table 8: Arguments for `GetCurrentConnections()`

Argument	Direction	relatedStateVariable
CurrentConnectionsList	OUT	CurrentConnections

2.4.3.2. Dependency on State (if any)

None

2.4.3.3. Effect on State

None

2.4.3.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
707	Operation Rejected	**GetCurrentConnections()** failed because the Remote UI Client has rejected the operation.

2.4.4. GetDeviceProfile

This action retrieves static information about the Remote UI Client remoting capabilities. The primary information returned by this action is the list of remoting protocols supported by the client.

2.4.4.1. Arguments

Table 9: Arguments for `GetDeviceProfile()`

Argument	Direction	relatedStateVariable
StaticDeviceInfo	OUT	DeviceProfile

2.4.4.2. Dependency on State (if any)

None

2.4.4.3. Effect on State (if any)

None

2.4.4.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
707	Operation Rejected	**GetDeviceProfile()** failed because the Remote UI Client has rejected the operation.

2.4.5. GetUIListing

This optional action retrieves the XML-formatted listing of user interfaces that are compatible with this client device. The listing is set with the **AddUIListing()** action.

2.4.5.1. Arguments

Table 10: Arguments for `GetUIListing()`

Argument	Direction	relatedStateVariable
`CompatibleUIList`	*OUT*	`A_ARG_TYPE_CompatibleUIs`

2.4.5.2. Dependency on State (if any)

None

2.4.5.3. Effect on State (if any)

None

2.4.5.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
707	Operation Rejected	**GetUIListing()** failed because the Remote UI Client has rejected the operation.

2.4.6. AddUIListing

This optional action allows a control point to inform a Remote UI client of available UIs that are compatible with its supported remoting protocols. In most cases, compatible UIs added to the listing will be displayed on the client device in the form of a menu entry of available applications.

- All UIs successfully added to the Remote UI client list with the **AddUIListing()** action are correspondingly returned by the **GetUIListing()** action.
- A URI corresponding to a UI already appearing in the UI list can be updated with new metadata by performing **AddUIListing()** with a new UI containing the same URI as the original, but possessing different metadata.

- A control point is not allowed to add (or remove) a UI of type *local* from the listing.
- The **AddUIListing()** action has no affect on the currently active user interfaces on the client device.

2.4.6.1. Arguments

Table 11: Arguments for `AddUIListing()`

Argument	Direction	relatedStateVariable
`InputUIList`	*IN*	`A_ARG_TYPE_CompatibleUIs`
`TimeToLive`	*OUT*	`A_ARG_TYPE_Int`

- ***TimeToLive*** is returned by the client device as a means of indicating the time, in seconds, that this list of UIs will be maintained in memory.

2.4.6.2. Dependency on State (if any)

2.4.6.3. Effect on State

None

2.4.6.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
707	Operation Rejected	**AddUIListing()** failed because the Remote UI Client has rejected the operation.
712	Invalid Input Argument	**AddUIListing()** failed because the *InputUIList* input argument is improperly formatted.

2.4.7. RemoveUIListing

This optional action allows a control point to remove one or more user experiences from the listing on a Remote UI device.

2.4.7.1. Arguments

Table 12: Arguments for `RemoveUIListing()`

Argument	Direction	relatedStateVariable
`RemoveUIList`	*IN*	`A_ARG_TYPE_String`

- ***RemoveUIList*** is composed of a comma-separated list of URIs. The Remote UI client searches the current UI listing and finds matches for the URIs in ***RemoveUIList***. All user interface entries possessing URIs that match the URIs in ***RemoveUIList*** are removed.
- Local user interfaces can not be removed from client compatible user interface listings.
- The **RemoveUIListing()** action has no affect on the currently active user interfaces on the client device.

2.4.7.2. Dependency on State (if any)

2.4.7.3. Effect on State

None

2.4.7.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
707	Operation Rejected	**RemoveUIListing()** failed because the Remote UI Client has rejected the operation.
712	Invalid Input Argument	**RemoveUIListing()** failed because the *RemoveUIList* input argument is improperly formatted.

2.4.8. DisplayMessage

This optional action displays a message on the Remote UI client. This action allows a wide-range of other UPnP devices (as control points) to send a notification to a remote UI client without understanding a specific remoting protocol (e.g. a washing machine sends “Laundry Ready” to a remote UI display).

- If this optional method is implemented, at least the data type *text/plain* must be supported.
- If the message type is *text/uri*, the content contains a URL pointing to the data.
- The complete list of supported data types can be queried on a given Remote UI client device by viewing the allowed value list in the service description.
- In order to prevent annoying messages from other devices, the manufacture should provide a way to disable this operation for specific devices (e.g. 'blacklist').
- **DisplayMessage()** fails (with error code 708) if the specified data type is not supported.
- **DisplayMessage()** fails (with error code 709) if the user has disabled this action.
- **DisplayMessage()** fails (with error code 710) if the client device does not allow the rendering of this message (e.g. higher-priority session is running).

2.4.8.1. Arguments

Table 13: Arguments for `DisplayMessage()`

Argument	Direction	relatedStateVariable
`MessageType`	*IN*	`A_ARG_TYPE_DisplayMessageType`
`Message`	*IN*	`A_ARG_TYPE_String`

2.4.8.2. Dependency on State (if any)

The remote UI client displays the message (or plays back the audio data). Depending on the UI , a message may be displayed inside a (pop-up) window, as a small icon activating the message when user selects it, inside the whole display or mixed into the current audio stream.

2.4.8.3. Effect on State

None

2.4.8.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
708	Unsupported Message Type	**DisplayMessage()** failed due to unsupported message type.
709	Disabled by user	**DisplayMessage()** failed because pop-up messages are currently disabled.
710	Busy state	**DisplayMessage()** failed because the device is in a state where additional messages cannot be displayed.

2.4.9. ProcessInput

This action allows control points to send user input to be processed just as if it was entered by a local user of the Remote UI client. Each call to **ProcessInput()** can contain one or more user input strings. The receiving Remote UI client device should process this user input just as if the user had pressed buttons directly on the local device itself.

For example: A UPnP enabled DVD player implements a Remote IO Client device that includes the **ProcessInput()** action. With this action, the DVD user can press buttons located on the device itself or on the infrared remote control to navigate thru menus, selection items, etc. Using the **ProcessInput()** action, a control point located on the user's PDA can discover and send user input to the DVD, mimicking the behavior of the DVD's infrared remote control.

2.4.9.1. Arguments

Table 14. Arguments for `ProcessInput()`

Argument	Direction	relatedStateVariable
InputDataType	*IN*	A_ARG_TYPE_InputDataType
InputData	*IN*	A_ARG_TYPE_String

2.4.9.2. Dependency on State (if any)

None

2.4.9.3. Effect on State

None

2.4.9.4. Errors

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
707	Operation Rejected	**ProcessInput()** failed because the Remote UI Client has rejected the operation.

2.4.10.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).

2.4.11.Relationships Between Actions

All actions defined have no specific relationship between them.

2.4.12.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 15: Common Error Codes

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
701	More Than One New Session	The action failed because more than one new session requested for connection is contained in the input argument.
702	Already Connected	The action failed because the remote UI client is already connected to this user interface.

errorCode	errorDescription	Description
703	UI Server Failure	The action failed because the out-of-band remoting protocol failed to establish a session due to a UI server failure.
704	Session Connection Timeout	The action failed because the out-of-band remoting protocol failed to establish a session in the time allotted..
705	*ConnectionUpdateID* Value Mismatch	The action failed because the value of *ConnectionsUpdateID* does not match the current value of *ConnectionsUpdateID* stored on the Remote UI client device.
706	Max Hold Capacity Exceeded	The action failed because it would have caused the number of on-hold sessions to exceed the capacity of the Remote UI client.
707	Operation Rejected	The Remote UI Client has rejected the operation.
708	Unsupported Message Type	The action failed due to unsupported message type.
709	Disabled by user	The action failed because pop-up messages are currently disabled.
710	Busy state	The action failed because the device is in a state where additional messages cannot be displayed.
711	Unknown UI	The action failed because one or more UIs specified did not correspond to an active or on-hold UI.
712	Invalid Input Argument	The action failed because one or more input arguments are improperly formatted.

3. Theory of Operation

3.1. Example Values of State Variables

3.1.1. A_ARG_TYPE_URI[1]

An *A_ARG_TYPE_URI* value is a string formatted as a URI. Client devices must be able to support values of *A_ARG_TYPE_URI* that are 1024 bytes in length. Client device support for longer values is optional. All *A_ARG_TYPE_URI* values are never URI escaped and must all be UTF-8 encoded.

3.1.1.1. UIs

UIs are *A_ARG_TYPE_URI* strings of the following form:

<***PI***>*://*<***SIP***>*[:*<***LPT***>*][/**AID**]*,

where:

PI: A Protocol identifier string. A short string that identifies a peer-to-peer remoting protocol, eg: *RDP*, *VNC*, *XRT*, etc.

SIP: An IP address of a server device.
The form of *SIP* is a true IPv4 or IPv6 internet address, not an address *name*.

LPT: A Server port number.

AID: An Application ID. A string that identifies a user interface or a remote-capable application. The value of *AID* can include a session ID.

The protocol name *local* is a reserved protocol name for user interfaces that are implemented by the Remote UI client itself. The protocol *local* must always be used with the IP address 127.0.0.1 and the port is never specified. For example a Remote UI client connected to "local://127.0.0.1/DvdBrowser" is connected to its own built-in DVD Browser application. The keyword *local* must always be used in lower case.

- Additionally, the user interface with value "local://127.0.0.1/null", all in lower case, is a reserved *UI* value that signifies that the Remote UI client is currently connected to a null local interface, a local user interface that is blank or has a message like "Please wait…" currently displayed on it.
- If a Remote UI client does not have any local user interface, the Remote UI client must implement the null user interface.
- Remote UI clients can implement more than one *local* interface.

*UI*s are strings of type *A_ARG_TYPE_URI***.** Remote UI client devices and control points must support *A_ARG_TYPE_URI* values that are at least 1024 bytes in length. Support for longer values is optional**.**

*UI*s must not be HTTP/URI escaped.

[1] *A_ARG_TYPE_URI* is not an actual Remote UI Client state variable. However, its format as described here (which is the same format of the Remote UI Server's *A_ARG_TYPE_URI*, which *is* an actual state variable for that service) is used to define other state variables that are used by Remote UI Clients.

*UI*s can contain spaces and tabs, but cannot begin with a white space character.

Example:
The following UI corresponds to a Remote UI-enabled DVD browser application available at IP address and port number *1.8.7.2:333.* The XRT2 remoting protocol is used in this case:

XRT2://1.8.7.2:333/DVDui

3.1.2. CurrentConnections

A *CurrentConnections* value is composed of a comma-separated list of all current Remote UI client sessions.

The first field of a *CurrentConnections* value must always contain the *ConnectionsUpdateID* value as the first comma-delimited field. A Remote UI client device increments the *ConnectionsUpdateID* value each time a **Connect()** or **Disconnect()** action is successfully completed. *ConnectionsUpdateID* rolls over to a value of one after it reaches the maximum value. The maximum value is *2147483647.*

All subsequent fields following the *ConnectionsUpdateID* are *A_ARG_TYPE_URI* values.

The first *A_ARG_TYPE_URI* value following *ConnectionsUpdateID* corresponds to the currently active user interface. If there is currently no active connection, a value corresponding to the Remote UI client default user interface, of the form "local://127.0.0.1/null" may appear in this field. If a Remote UI client does not possess a local user interface, it must implement a null user interface.

All of the subsequent URIs enumerate each user interface placed on hold. The URI in the list commas-separated list corresponds to the first session established.

Example:

```
5604,
XRT://1.23.345.1/My_Music_Player0xa12c67
local://127.4.6.1/null,
VNC://1.23.345.2/My_Photo_Viewer,
RDP://1.23.345.3/Super_Chess
```

3.1.3. A_ARG_TYPE_CompatibleUIs

The value of an *A_ARG_TYPE_CompatibleUIs* is an XML block corresponding to a list of UIs. Remote UI client devices must be able to support values of *A_ARG_TYPE_CompatibleUIs* that are 10k bytes in length. Remote UI client device support for longer values is optional.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<uilist xmlns="urn:schemas-upnp-org:remoteui:uilist-1-0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:schemas-
upnp-org:remoteui:uilist-1-0 CompatibleUIs.xsd">
  <ui>
    <uiID>6789-568</uiID>
    <name>DVD Browser</name>
    <protocol shortName="VNC">
      <uri>VNC://1.8.7.2:5920</uri>
    </protocol>
  </ui>
  <ui>
    <uiID>6789-569</uiID>
```

```
    <name>DVD Recording Setup</name>
    <protocol shortName="XRT2">
      <uri>XRT2://1.8.7.2:333/DVDRec/fdc4510ae512</uri>
    </protocol>
  </ui>
</uilist>
```

See the *RemoteUIServer Services Document* for a detailed description of the *A_ARG_TYPE_CompatibleUIs* elements.

All URIs appearing in the *A_ARG_TYPE_CompatibleUIs* listing must be unique.

Section 5 contains the XSD schema that can be used to validate *A_ARG_TYPE_CompatibleUIs* values.

3.1.4. DeviceProfile

DeviceProfile values are XML-formatted strings used by the client device to represents the list of all remoting protocols supported by the Remote UI client device. The format of the device profile is UTF-8 encoded XML. See section 4 for the XSD schema that formally defines the format of *DeviceProfile* values.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<deviceprofile xmlns="urn:schemas-upnp-org:remoteui:devprofile-1-0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:schemas-
upnp-org:remoteui:devprofile-1-0 DeviceProfile.xsd">
  <maxHoldUI>5</maxHoldUI>
  <protocol shortName="LRDP">
    <protocolInfo>LRDP:image 1500 UDP beep:sendonly</protocolInfo>
  </protocol>
  <protocol shortName="XHT">
    <protocolInfo> (opaque) </protocolInfo>
  </protocol>
  <protocol shortName="RDP"/>
  <protocol shortName="XRT2">
<protocolInfo>version=2.1,displayWidth=640,displayHeight=480,imageEncoding=JPEG&amp;PN
G,serverVolumeControl=TRUE,videoViewPortRequired=TRUE</protocolInfo>
  </protocol>
</deviceprofile>
```

A_ARG_TYPE_DeviceProfile values are validated with the XSD schema in section 5.

<MaxHoldUI>

This optional element defines how many user interfaces can be put on hold or in the background. If the Remote UI client does not support placing user interfaces on hold, the **<MaxHoldUI>** value is set to *0*.

The default value of **<MaxHoldUI>** is *0*.

<protocol>

This required tag contains all the information needed for a specific UI remoting protocol. Multiple child **<protocol>** elements may be used to indicate support for more than one remoting protocol.

The **<protocol>** tag must define a string value for the required *shortName* attribute. Values for *shortName* corresponding to known Remote UI protocols appear in table 7 of the *UPnP*

RemoteUIServer:1 service document. Implementations employing one or more of the protocols listed in table 7 must use the *shortName* string values as they are shown in the table (all capital letters).

Implementations may also use *shortName* to expose support for vendor-specific protocols. Vendor-defined *shortName* values may use any combination of upper or lower case letters. Short name values must be UTF-8 encoded and no longer than 256 bytes.

<protocolInfo>

The optional **<protocolInfo>** tag contains a block of data that is specific to a given remoting protocol. This block of data may contain information that can help in establishing preferences or compatibility between a Remote IU server and a Remote UI client.

Some protocols negotiate all of the session parameters out-of-band upon establishment of the remoting session. **RDP** for example, requires no additional compatibility criteria to be provided by UPnP Remote UI. In these cases the **<protocolInfo>** block is not needed.

3.2. Remote UI Scenarios for the Basic DCP

3.2.1. Connect, Disconnect and GetCurrentConnections

The most basic scenario enabled by UPnP Remote UI, is connecting a Remote UI Client to a UI served by a Remote UI Server, and subsequently disconnecting this Remote UI Client from that UI.

To establish a connection between a Remote UI Client and a UI, two pieces of information need to be passed when calling that Client's **Connect()** action: a *ConnectionsUpdateID* and the *URI* of the UI to connect to. The URI can be retrieved using the **GetCompatibleUIs()** action on a Remote UI Server. The value of *ConnectionsUpdateID must* be equal to the current value of *ConnectionsUpdateID* on the Remote UI Client, and is therefore best retrieved by calling **GetCurrentConnections()** on the Remote UI Client shortly before trying to establish the connection. Upon calling **Connect()** on a Remote UI Client, the Remote UI Client will use an out-of-band mechanism (as provided by the remote UI protocol identified by the specified URI) to connect to the UI.

A connection between a Remote UI Client and a UI can be terminated by calling that Client's **Disconnect()** action. Again, two pieces of information need to be passed: a *ConnectionsUpdateID* value equal to the current value of *ConnectionsUpdateID* on the Client and the URI of the connected UI. Both can be retrieved by calling **GetCurrentConnections()** on the Remote UI Client.

3.2.2. Add, Get and Remove UI Listings

Users of Remote UI Client devices possessing interactive user interfaces should be given a means to select among listings of UIs that are compatible with and available to the client device. One way of doing this is to have a Remote UI Client store a UPnP-accessible list of UIs with which it is compatible. This UI listing can be displayed on the Remote UI Client with a local shell application or by remoting the UI of an external shell application directly to the Client..

UIs can be added to and removed from this list on a Remote UI Client using the Client's **AddUIListing()** and **RemoteUIListing()** actions. **GetUIListing()** returns the current list of UIs.

3.2.3. Display Message

To display a message on a Remote UI Client without setting up a connection, a Client's **DisplayMessage()** action can be used. This action is meant to be used principally for notifications. User interaction with a UI over a longer period of time is better handled by connecting to that UI using the Client's **Connect()** action.

3.2.4. Process Input

User input can be send to a Remote UI Client using the Client's **ProcessInput()** action. The Remote UI Client device treats input received through its **ProcessInput()** action as though it was input locally.

3.3. Remote UI Scenarios for the Advanced DCP

3.3.1. Mirror

A user may desire to make a UI session currently running on one Remote UI Client device available on another Remote UI Client, for example to share the experience. This process is referred to as *mirroring* the original UI on another Remote UI Client device.

To mirror a UI that is currently connected to Remote UI Client device *A* on Remote UI Client device *B*, a Remote UI Client control point must first fetch the information associated with the UI targeted for mirroring by calling the **GetCurrentConnections()** action on Remote UI Client *A*. The Remote UI Client control point then calls **Connect()** on Remote UI Client *B* using the URI information obtained from client device *A*. If multiple connections to the same UI are not supported or the specified UI on client *A* is not compatible with Remote UI Client *B*, the **Connect()** action on Remote UI Client *B* fails.

Note that 'forking' UIs, i.e. UIs for which the Remote UI Server has returned a **<fork>** value of *true* do not support mirroring. Connecting to a forking UI will always yield a new UI session, independently of the UI sessions that may have been previously established by connecting to the forking UI. A forking UI however, may spawn a non-forking UI which can be mirrored. See the *RemoteUIServices* document for more information on forking UIs.

3.3.2. Move

A user may also desire to move a UI running on one Remote UI Client device to another Remote UI Client and continue using the UI session. This process is referred to as *moving* the original UI to another Remote UI Client device.

To move a UI that is currently connected to Remote UI Client device *A* to Remote UI Client *B*, a Remote UI Client control point must first fetch the information associated with the UI targeted for moving by calling **GetCurrentConnections()** on Remote UI Client *A*. The control point then calls **GetCurrentConnections()** on Remote UI Client *B* to get its current *ConnectionsUpdateID* value. It then calls **Disconnect()** on Remote UI Client *A* to disconnect the UI targeted for moving. The control point finally invokes the **Connect()** action on Remote UI Client *B* using the *ConnectionsUpdateID* value for client *B*, along with the targeted UI information obtained from the **GetCurrentConnections()** action on client *A*.

Moving a UI for which the Remote UI Server returns a **<lifetime>** of *0* is likely to fail, because the Remote UI Server may destroy that UI as soon as it detects that no clients are connected to it. For UIs that enable it, the UI post-disconnect lifetime may be extended by calling **SetUILifetime()** on the Remote UI Server device.

Note that 'forking' UIs, i.e. UIs for which the Remote UI Server has returned a **<fork>** value of *true* do not support moving. Connecting to a forking UI will always yield a new UI session, rendering it impossible to connect to a forking UI for the purpose of continuing an existing session. A forking UI however, may spawn a non-forking UI which can be moved. See the *RemoteUIServices* document for more information on forking UIs.

3.3.3. Reconnect

The process of connecting to an existing UI session, that was either orphaned or deliberately disconnected and kept alive at the request of a user for the purpose of continuing to interact with it at a later time is referred to as *reconnecting* to that UI.

To reconnect a UI to a Remote UI Client, a Remote UI Client control point must first fetch the URI and information associated with the targeted UI by calling the **GetCompatibleUIs()** action on the Remote UI Server that hosts the UI. Using the information obtained from the server, the control point then calls **Connect()** on a designated Remote UI Client device to reconnect to the original, targeted UI.

Reconnecting to a UI for which the Remote UI Server returns a **<lifetime>** of *0* is likely to fail, because the Remote UI Server may destroy that UI as soon as it detects that no clients are connected to it. For UIs that enable it, the UI post-disconnect lifetime may be extended by calling **SetUILifetime()** on the Remote UI Server device.

3.4. Types of Client Devices

3.4.1. Autonomous Remote UI Clients

Remote UI clients may possess a fully functional, local user interface of their own. and may not need a remoted user interface except for providing additional experiences. Client devices of this type include a local user interface for selecting which remoted UIs to view and are generally self-managed. These Remote UI clients will support the optional Add, Get and RemoveUIListing() actions, allowing Remote UI Client control points on the network to forward listings of compatible user interfaces. Management of the user interface connection by external client control points may not be allowed since these Remote UI client devices are generally in full control of their own user interface. Examples of these device types include: Desktop and mobile PC's, Tablets, PDA's, High-end LCD remote controls.

3.4.2. Fully Remoted Remote UI Clients

This category of Remote UI clients don't possess a local user interface and are dependent on other devices on the network to provide UIs. If multiple, available interfaces are compatible with this device, a UI needed to allow the user to select from the UI listings must be remoted from another device. These Remote UI clients will support only the required Remote UI Client device actions like Connect(), etc., allowing external Remote UI Client control points on the network to take control of the device. In some cases the remoted UI selection interface may be the first connection, allowing the user to switch to other UIs. Examples of these device types include: PVR set-top boxes, DVD players, low-end LCD remote controls, Digital Media Adapters, TV's and etc.

4. DeviceProfile XSD Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" id="deviceprofile">
  <xs:element name="deviceprofile">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="maxHoldUI" type="xs:unsignedInt" default="0" minOccurs="0"/>
        <xs:element name="protocol" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="protocolInfo" type="xs:anyType" nillable="true" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="shortName" type="xs:string" use="required" form="unqualified"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5. A_ARG_TYPE_CompatibleUIs XSD Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:schemas-upnp-org:remoteui:uilist-1-0"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" id="uilist">
 <xs:element name="uilist">
  <xs:complexType>
   <xs:sequence maxOccurs="unbounded">
    <xs:element name="ui">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="uiID" type="xs:string"/>
       <xs:element name="name" type="xs:string"/>
       <xs:element name="description" type="xs:string" minOccurs="0"/>
       <xs:element name="iconList" minOccurs="0">
        <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
          <xs:element name="icon">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="mimetype" type="xs:string"/>
             <xs:element name="width" type="xs:positiveInteger"/>
             <xs:element name="height" type="xs:positiveInteger"/>
             <xs:element name="depth" type="xs:positiveInteger"/>
             <xs:element name="url" type="xs:anyURI"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="fork" type="xs:boolean" default="false" minOccurs="0"/>
       <xs:element name="lifetime" type="xs:integer" default="-1" minOccurs="0"/>
       <xs:element name="protocol" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="uri" type="xs:anyURI" nillable="false"
maxOccurs="unbounded"/>
          <xs:element name="protocolInfo" type="xs:anyType" nillable="true" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="shortName" type="xs:string" use="required" form="unqualified"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

6. XML Service Description

```
<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
 <specVersion>
  <major>1</major>
  <minor>0</minor>
 </specVersion>
 <actionList>
  <action>
   <name>AddUIListing</name>
   <argumentList>
    <argument>
     <name>InputUIList</name>
     <direction>in</direction>
     <relatedStateVariable>A_ARG_TYPE_CompatibleUIs</relatedStateVariable>
    </argument>
    <argument>
     <name>TimeToLive</name>
     <direction>out</direction>
     <relatedStateVariable>A_ARG_TYPE_Int</relatedStateVariable>
    </argument>
   </argumentList>
  </action>
  <action>
   <name>Connect</name>
   <argumentList>
    <argument>
     <name>RequestedConnections</name>
     <direction>in</direction>
     <relatedStateVariable>CurrentConnections</relatedStateVariable>
    </argument>
    <argument>
     <name>CurrentConnectionsList</name>
     <direction>out</direction>
     <relatedStateVariable>CurrentConnections</relatedStateVariable>
    </argument>
   </argumentList>
  </action>
  <action>
   <name>Disconnect</name>
   <argumentList>
    <argument>
     <name>RequestedDisconnects</name>
     <direction>in</direction>
     <relatedStateVariable>CurrentConnections</relatedStateVariable>
    </argument>
    <argument>
     <name>CurrentConnectionsList</name>
     <direction>out</direction>
     <relatedStateVariable>CurrentConnections</relatedStateVariable>
    </argument>
   </argumentList>
  </action>
```

```
<action>
  <name>DisplayMessage</name>
  <argumentList>
    <argument>
      <name>MessageType</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_DisplayMessageType</relatedStateVariable>
    </argument>
    <argument>
      <name>Message</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_String</relatedStateVariable>
    </argument>
  </argumentList>
</action>
<action>
  <name>GetCurrentConnections</name>
  <argumentList>
    <argument>
      <name>CurrentConnectionsList</name>
      <direction>out</direction>
      <relatedStateVariable>CurrentConnections</relatedStateVariable>
    </argument>
  </argumentList>
</action>
<action>
  <name>GetDeviceProfile</name>
  <argumentList>
    <argument>
      <name>StaticDeviceInfo</name>
      <direction>out</direction>
      <relatedStateVariable>DeviceProfile</relatedStateVariable>
    </argument>
  </argumentList>
</action>
<action>
  <name>GetUIListing</name>
  <argumentList>
    <argument>
      <name>CompatibleUIList</name>
      <direction>out</direction>
      <relatedStateVariable>A_ARG_TYPE_CompatibleUIs</relatedStateVariable>
    </argument>
  </argumentList>
</action>
<action>
  <name>ProcessInput</name>
  <argumentList>
    <argument>
      <name>InputDataType</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_InputDataType</relatedStateVariable>
    </argument>
    <argument>
      <name>InputData</name>
```

```
        <direction>in</direction>
        <relatedStateVariable>A_ARG_TYPE_String</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
  <action>
    <name>RemoveUIListing</name>
    <argumentList>
      <argument>
        <name>RemoveUIList</name>
        <direction>in</direction>
        <relatedStateVariable>A_ARG_TYPE_String</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
</actionList>
<serviceStateTable>
  <stateVariable sendEvents="no">
    <name>A_ARG_TYPE_CompatibleUIs</name>
    <dataType>string</dataType>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>A_ARG_TYPE_URI</name>
    <dataType>string</dataType>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>CurrentConnections</name>
    <dataType>string</dataType>
  </stateVariable>
  <stateVariable sendEvents="yes">
    <name>CompatibleUIsUpdateIDEvent</name>
    <dataType>i4</dataType>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>A_ARG_TYPE_InputDataType</name>
    <dataType>string</dataType>
    <allowedValueList>
      <allowedValue>ASCII</allowedValue>
      <allowedValue>UNICODE</allowedValue>
      <allowedValue>ISO10646</allowedValue>
      <allowedValue>ISO8859</allowedValue>
    </allowedValueList>
  </stateVariable>
  <stateVariable sendEvents="yes">
    <name>CurrentConnectionsEvent</name>
    <dataType>string</dataType>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>A_ARG_TYPE_Int</name>
    <dataType>i4</dataType>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>DeviceProfile</name>
    <dataType>string</dataType>
  </stateVariable>
```

```
    <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_String</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_DisplayMessageType</name>
      <dataType>string</dataType>
      <allowedValueList>
        <allowedValue>text/plain</allowedValue>
      </allowedValueList>
    </stateVariable>
  </serviceStateTable>
</scpd>
```