InputConfig:1 Service

For UPnP Version 1.0
Status: Standardized DCP (SDCP)
Date: March 22, 2011
Document Version: 1.0
Service Template Version: 2.00

Authors[1]	Company
Yoshiki Nishikawa	NTT
David Moreo	Telecom Italia
Enrico Grosso	Telecom Italia
Massimo Messore	Telecom Italia
Mahfuzur Rahman(editor)	Samsung
Mayuresh Patil	Samsung
Jeyoung Maeng(editor)	Samsung
Jooyeol Lee (Chair)	Samsung
Yu Zhu	Huawei
Andreas Kraft	Deutsche Telekom
Vivien Helmut	Deutsche Telekom

[1] Note: The UPnP Forum in no way guarantees the accuracy or completeness of this author list and in no way implies any rights for or support from those members listed. This list is not the specifications' contributor list that is kept on the UPnP Forum's website.

Contents

List of Tables

List of Figures

1 Overview and Scope

This service definition is compliant with the UPnP Device Architecture version 1.0. It defines a service type referred to herein as *InputConfig* service.

This service type enables an in-band configuration mechanism between input devices which have embedded *InputConfig* service.

1.1 Introduction

The *InputConfig* service enables modeling of input data transferring capabilities of Input devices, and binding of those capabilities between devices. Each device should have an instance of the *InputConfig* service in order to send or receive input data, according to the architecture defined in this service.

A role is assigned to each Input device to define whether the device is the source or the sink of the input data transport. A data type is associated with an input connection between the source and the sink. . For example, if the “sender” role and the “text” type are assigned to an Input device, the device will be ready to send input data of text format to the sink device.

The *InputConfig* service is generic enough to properly abstract different kinds of input data format and transport mechanism, such as USB HID compliant input data over TCP/IP based transport protocol or IrDA based transport.

This service provides Control Points with the following functionality:

- Perform capability matching between input devices.
- Setup and teardown connections between devices.
- Change assigned roles and input type between devices.
- Notify changes of the input type that the receiver requests.

This service does not address:

- Data format used to encode the input data.
- Transport protocol used to input data session.

1.2 Notation

- In this document, features are described as Required, Recommended, or Optional as follows:

  The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this specification are to be interpreted as described in [RFC 2119].

  In addition, the following keywords are used in this specification:

  PROHIBITED – The definition or behavior is an absolute prohibition of this specification. Opposite of REQUIRED.

  CONDITIONALLY REQUIRED – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is REQUIRED, otherwise it is PROHIBITED.

  CONDITIONALLY OPTIONAL – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is OPTIONAL, otherwise it is PROHIBITED.

  These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

- Strings that are to be taken literally are enclosed in "double quotes".
- Words that are emphasized are printed in *italic*.
- Keywords that are defined by the UPnP Working Committee are printed using the *forum* character style.
- Keywords that are defined by the UPnP Device Architecture are printed using the **arch** character style.
- A double colon delimiter, "::", signifies a hierarchical parent-child (parent::child) relationship between the two objects separated by the double colon. This delimiter is used in multiple contexts, for example: Service::Action(), Action()::Argument, parentProperty::childProperty.

1.2.1 Data Types

This specification uses data type definitions from two different sources. The UPnP Device Architecture defined data types are used to define state variable and action argument data types [DEVICE]. The XML Schema namespace is used to define property data types [XML SCHEMA-2].

For UPnP Device Architecture defined Boolean data types, it is strongly RECOMMENDED to use the value "**0**" for false, and the value "**1**" for true. The values "**true**", "**yes**", "**false**", or "**no**" MAY also be used but are NOT RECOMMENDED. The values "**yes**" and "**no**" are deprecated and MUST NOT be sent out by devices but MUST be accepted on input.

For XML Schema defined Boolean data types, it is strongly RECOMMENDED to use the value "*0*" for false, and the value "*1*" for true. The values "*true*", "*yes*", "*false*", or "*no*" MAY also be used but are NOT RECOMMENDED. The values "*yes*" and "*no*" are deprecated and MUST NOT be sent out by devices but MUST be accepted on input.

1.3 Vendor-defined Extensions

Whenever vendors create additional vendor-defined state variables, actions or properties, their assigned names and XML representation MUST follow the naming conventions and XML rules as specified in [DEVICE], Section 2.5, "Description: Non-standard vendor extensions".

1.4 References

1.4.1 Normative References

This section lists the normative references used in this specification and includes the tag inside square brackets that is used for each such reference:

[DEVICE] – UPnP Device Architecture, version 1.0, UPnP Forum, June 13, 2000.
Available at: http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0-20000613.htm.
Latest version available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf.

[ISO 8601] – Data elements and interchange formats – Information interchange -- Representation of dates and times, International Standards Organization, December 21, 2000.
Available at: ISO 8601:2000.

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

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

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

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

1.4.2 Informative References

This section lists the informative references that are provided as information in helping understand this specification:

[PHONEARCH] – TelephonyArchitecture:1, UPnP Forum, March 22, 2011.
Available at: http://www.upnp.org/specs/phone/UPnP-phone-TelephonyArchitecture-v1-20110322.pdf.
Latest version available at: http://www.upnp.org/specs/phone/UPnP-phone-TelephonyArchitecture-v1.pdf.

[USBSPEC] – *Universal Serial Bus (USB) Specification,Version 1.0*, USB Implementation Forum, Inc., November 12, 2008.
Available at: http://www.usb.org/developers/docs/usb_30_spec_052109.zip
Latest version available at: http://www.usb.org/developers/docs/usb_30_spec_052109.zip

[USBHID] – *Device Class Definition for Human Interface Devices (HID)*, USB Implementation Forum, Inc., June 27, 2001.
Available at: http://www.usb.org/developers/devclass_docs/HID1_11.pdf
Latest version available at: http://www.usb.org/developers/devclass_docs/HID1_11.pdf

[USBHUT] – *HID Usage Tables*, UPnP Forum, October 28, 2004.
Available at: http://www.usb.org/developers/devclass_docs/Hut1_12.pdf
Latest version available at: http://www.usb.org/developers/devclass_docs/Hut1_12.pdf

[BTHID] – *Human Interface Device (HID) Profile*, Bluetooth SIG, May 22, 2005.
Available at: http://www.bluetooth.com/NR/rdonlyres/FEF51323-1EEC-49D6-A78F-ABF2277C202C/980/HID_SPEC_V10.pdf
Latest version available at: http://www.bluetooth.com/NR/rdonlyres/FEF51323-1EEC-49D6-A78F-ABF2277C202C/980/HID_SPEC_V10.pdf

[RFC4347] – *Datagram Transport Layer Security version 1.2*, IETF, October 7, 2009.
Available at: http://tools.ietf.org/html/draft-ietf-tls-rfc4347-bis-03
Latest version available at: http://tools.ietf.org/html/draft-ietf-tls-rfc4347-bis-03

2 Service Modeling Definitions (Normative)

2.1 Service Type

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

urn:schemas-upnp-org:service:*InputConfig:1*

InputConfig:1 service is used herein to refer to this service type.

2.2 Terms and Abbreviations

2.2.1 Abbreviations

Table 2-1: Abbreviations

Abbreviation	Description
TC	Telephony Client
TS	Telephony Server
TelCP	Telephony Control Point
ID	Input Device
ICP	InputConfig Control Point
ICS	InputConfig Service

2.2.2 Terms

2.2.2.1 Capability

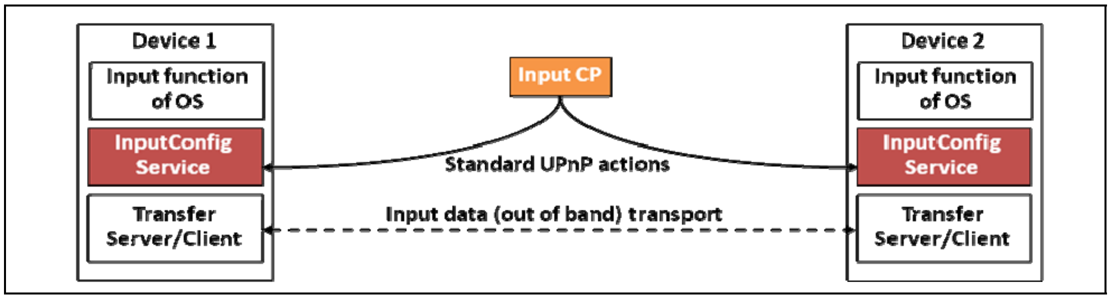
The capability is a general term used in the specification to describe a data structure that includes detailed information of input device to establish an input session. The capability includes possible roles, input mode, and the transport protocols supported by the Input device.

2.2.2.2 Role (sender/receiver)

The role is a property of the input device that represents the behavior of the device when the device participates in an input session. There are two types of roles defined in the *InputConfig* service. The “sender” role means that the device is responsible to send input data to the sink device. The “receiver” role means that the device is responsible to receive the input data.

2.3 Service Architecture

This service provides features for an UPnP Control Point to present and configure the input capability through UPnP standard. The device that has been assigned the role of a sender should be in charge of sending the input data and it should pair off with a device which is assigned the role of a receiver which will receive the sender's input data. In this case, both of the devices MUST embed *InputConfig* service.



The above architecture describes a 3-box model which is comprised of Input Control Point (ICP) and input devices that include *InputConfig* service (ICS). It is possible to embed an Input Control Point either in the sender or in the receiver device because the ICP is a logical entity that can be resided in all devices. Even when the input device contains the ICP, the sequence of configuration does not need to be changed because the ICP can replace the actions required for configuration to internal process.

The ICP retrieves input capability information of the devices, invoking an action exposed by the *InputConfig* service (ICS). Matching capabilities of the devices, the ICP selects the appropriate capability which includes role of the device, input type, and the protocol for the exchange of the input data and configure the devices using the capabilities.

The actual transport of input data occurs in an out- of- band mechanism. The Transfer server/client should be in charge of this operation and the *InputConfig* service (ICS) should have an internal interface to setup the Transfer server/client. The detailed description of this component is out of scope of this document.

The input data transferred between the Transfer server and client is delivered to the input function of the operating system in the input device as if the data is a kind of local input. However the detailed description of this component is also out of scope of this document.

Optionally, the *InputConfig* service (ICS) supports the establishment of a secure channel to transport the input data to protect sensitive information such as credit card number, private id, and password etc. The secure channel can be easily established by the TLS or DTLS mechanism which are widely used to establish a secure channel in the web area.

2.4 State Variables

The state variables represent the capability of the input device and the status of the device and configuration value.

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

2.4.1 State Variable Overview

Table 2-2: State Variables

Variable Name	R/O[1]	Data Type	Reference
DeviceInputCapability	*R*	**string**(XML fragment)	See Section 2.4.2
InputConnectionList	*R*	**string**(XML fragment)	See Section 2.4.3
RequiredInputType	*R*	**string**	See Section 2.4.4
A_ARG_TYPE_ConnectionInfo	*R*	**string**(XML fragment)	See Section 2.4.5
A_ARG_TYPE_InputCapability	*R*	**string**(XML fragment)	See Section 2.4.6
A_ARG_TYPE_DeviceInfo	*R*	**string**(XML fragment)	See Section 2.4.7
A_ARG_TYPE_InputSessionID	*R*	**ui4**	See Section 2.4.8
A_ARG_TYPE_MultiInputMode	*R*	**string**	See Section 2.4.9

[1] *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*).

2.4.2 *DeviceInputCapability*

This state variable contains the entire set of input capabilities of an input device which are used to setup an input session. The *DeviceInputCapability* state variable contains the value of the feasible roles, input mode, transport protocols, and the input data format a device supports.

The value of this state variable can be updated according to the change of current capability of the input device. Once updated, the state variable sends events to the interested ICPs. For example, if the device protects the data channel using DTLS and it decides not to use any protection mechanism, the input device may disable the DTLS function and change the corresponding entry in the *DeviceInputCapability*. And then the device notifies this change of the *DeviceInputCapability* to the interested ICPs. Therefore the *InputConfig* service MUST monitor whether the capability of the input device (receiver) has been changed.

The Control Point selects the best combination of these capabilities to configure input devices. Note that this state variable MUST not be changed by the Control Point. Therefore, the Control Point can only read this state variable and select the *InputCapability* argument for an input session.

The value of the report descriptor in this state variable should be encoded by the BASE64 defined in RFC1521. As the report descriptor is composed of binary code, the report descriptor should be encoded differently as compared to the other texts to be delivered through UPnP action and xml document.

If an Input Device supports a channel protection mechanism such as TLS and DTLS, it should represent the tag related to the mechanism in the *DeviceInputCapability*. TLS is used to protect the control channel of the input data and DTLS is used to protect the interrupt channel which delivers the actual input data. DTLS is almost similar to TLS, but it works on top of UDP transport protocol whereas TLS works on top of TCP. UDP use simple transmission model without implicit hand-shaking dialogues to guarantee reliability, ordering, or data integrity. To establish a TLS session between two devices, a group of sequential messages are exchanged between the devices participating in the session. For this reason, TLS does not work on top of UDP. To overcome the shortcomings of the TLS and to support establishment of a secure channel on top UDP protocol, IETF has defined DTLS and this specification adopt it for channel protection.

2.4.2.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *DeviceInputCapability* in the XML namespace "`urn:schemas-upnp-org:phone:ics`" which is located at "`http://www.upnp.org/schemas/phone/ics.xsd`".

2.4.2.2 Description of fields in the *DeviceInputCapability* structure

```
<?xml version="1.0" encoding="utf-8"?>
<ics:deviceInputCapability
   xmlns:ics="urn:schemas-upnp-org:phone:ics"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:phone:ics
   http://www.upnp.org/schemas/phone/ics.xsd">
   <capability role="sender">
      <inputModeList>
         <inputMode>
            <inputDeviceType>simple type</inputDeviceType>
            <inputDeviceSubClass>USB device class</inputDeviceSubClass>
            <inputDataformat type= "USB_RD">
               <reportDescriptor>
                  USB report descriptor
               </reportDescriptor>
            </inputDataformat>
            <transportProtocol>BT_HID_REQUEST</transportProtocol>
            <supportSecureChannel>
               <TLS>True or False</TLS>
               <DTLS>True or False</DTLS>
            </supportSecureChannel>
         </inputMode>
      </inputModeList>
   </capability>
   <capability role="receiver">
      <inputModeList>
         <inputMode>
            <inputDeviceType>simple type</inputDeviceType>
            <inputDeviceSubClass>USB device class</inputDeviceSubClass>
            <inputDataformat type= "USB_RD">
               <reportDescriptor>
                  USB report descriptor
               </reportDescriptor>
            </inputDataformat>
            <transportProtocol>BT_HID_REQUEST</transportProtocol>
            <supportSecureChannel>true or False</supportSecureChannel>
         </inputMode>
      </inputModeList>
   </capability>
</ics:deviceInputCapability>
```

`<xml>`
   REQUIRED. Case sensitive.

`<deviceInputCapability>`
REQUIRED. MUST include a namespace declaration for the InputConfig service deviceInputCapability Schema ("urn:schemas-upnp-org:phone:ics"). MUST include at least one of the following elements.

`<capability>`
REQUIRED. Indicates the required parameters to setup an input session between two input devices.

`role`
REQUIRED. xsd:string, Indicates the role of an Input device. For example, a "sender" role means that this Input device can send input data to its pair.

`<inputModeList>`
REQUIRED. Indicates a list of inputMode. It MUST include one or more of the following elements. The Input device MUST support the input mode which is mentioned in this description.

`<inputMode>`
REQUIRED. Indicates a set of parameters supported by the input device. It includes the following elements.

`<inputDeviceType>`
REQUIRED. Indicates the simple type of the input device. This element can have a value such as Keyboard, Keypad, Remocon, and Mouse etc.

`<inputDeviceSubClass>`
OPTIONAL. Indicates the device subclass defined in USB HID specification. If an input device supports the USB data format this element gives information about the receiver's USB device driver.

`<inputDataformat>`
REQUIRED. Indicates the format of the input data. The format of the input data is negotiated between the sender and the receiver device. Currently this version of the InputConfig service supports only one type of data format that is USB report descriptor.

`type`
REQUIRED. xsd:string, Indicates the type of data format for the input. The meaning of the format is the encoding type or packetizing type of the input data. Basically the type should have the USB_RD as a default value. If the vendor wants to adopt a new type of the input data format, the value of this element can be defined by the vendor.

`<reportDescriptor>`
REQUIRED. Indicates the report descriptor of the input data only if the value of the inputDataformat@type is USB_RD. The report descriptor should comply with the USB report descriptor standard.

`<transportProtocol>`
REQUIRED. Indicates a feasible transport protocol for the input data. Basically this element should have BT_HID_REQUEST as the default value. If the vendor wants to adopt a new transport protocol, the value of this element can be defined by the vendor.

`<supportSecureChannel>`
OPTIONAL. Indicates whether the input device supports a method to protect the input data stream. It has the following elements:

`<controlChannel>`
OPTIONAL. Indicates the mechanism to protect the control channel. The default value of element is TLS. If there is no <controlChannel> element under the <supportSecureChannel> element, it indicates that the input device does not support control channel protection.

`<dataChannel>`
OPTIONAL. Indicates the mechanism to protect the data channel. The default value of element is DTLS. If there is no <dataChannel> element under the <supportSecureChannel> element, then it indicates that the input device does not support data channel protection.

Note: Additional elements or attributes may also be present, for example, defined by individual vendors or future versions of the *InputConfig* service specification. Consequently, a Control Point must gracefully ignore any additional elements or attributes that it does not understand.

2.4.3 *InputConnectionList*

This state variable contains information related to the sessions that are currently managed by a eceiver.

This variable is updated by the *SetInputSession()* or the *StartInputSession()* actions. When the *SetInputSession()* is invoked, the input device prepares the session and generates a new session id to add it to the *InputConnectionList*.

2.4.3.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *InputConnectionList* in the XML namespace "`urn:schemas-upnp-org:phone:ics`" which is located at "`http://www.upnp.org/schemas/phone/ics.xsd`".

2.4.3.2 Description of fields in the *InputConnectionList* structure

```
<?xml version="1.0" encoding="utf-8"?>
<ics:inputConnectionList
   xmlns:ics="urn:schemas-upnp-org:phone:ics"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:phone:ics
    http://www.upnp.org/schemas/phone/ics.xsd">
   <multiInputMode>2</multiInputMode>
   <peerDeviceList>
      <peerDevice uuid="6E09886B-DC6E-439F-82D1-7CCAC7F4E3B4">
         <sessionList>
            <session localSessionID="1">
               <localDeviceRole>Sender</localDeviceRole>
               <inputType>TEXT</inputType>
               <sessionStatus>Active</sessionStatus>
               <peerSessionID>7</peerSessionID>
            </session>
         </sessionList>
      </peerDevice>
      <peerDevice uuid="7E09886B-DC6E-439F-82D1-7CCAC7F4E3B5">
         <sessionList>
            <session localSessionID="2">
               <localDeviceRole>Reciever</localDeviceRole>
               <inputType>coodinate</inputType>
               <sessionStatus>Idle</sessionStatus>
               <peerSessionID>6</peerSessionID>
            </session>
         </sessionList>
      </peerDevice>
   </peerDeviceList>
</ics:inputConnectionList>
```

`<xml>`
REQUIRED. Case sensitive.

`<inputConnectionList>`
REQUIRED. MUST include a namespace declaration for the InputConfig service inputConnectionList Schema ("urn:schemas-upnp-org:phone:ics"). MUST include one or more of the following elements. This namespace defines the following elements and attributes:

`<multiInputMode>`
REQUIRED. Indicates the multi input mode which means a rule to deal with input datum from multiple input devices. This element can have three values 1,2 and 3. The detail meaning of these value representing multi input mode is described in section 2.4.9

`<peerDeviceList>`
REQUIRED. Indicates the list which includes information of one or more peer devices.

`<peerDevice>`
REQUIRED. Indicates the information of peer devices which are connected with the local device that is currently controlled by a control point.

uuid
REQUIRED. xsd:string, Indicate the uuid of peer device.

`<sessionList>`
REQUIRED. Indicates the list which includes information of one or more sessions. All the session is connected to the specific peer device which uuid is described in the peerDevice element including sessionList element.

`<session>`
REQUIRED. Indicates the session information between the local device and the peer.

localSessionID
REQUIRED. xsd:string, Indicate the session ID assigned by the local device.

`<localDeviceRole>`
REQUIRED. Indicates the role of the local device.

`<inputType>`
REQUIRED. Indicates the input type of this session.

`<sessionStatus>`
REQUIRED. Indicates the status of this session. This element can have the value of either ACTIVE or IDLE.

`<peerSessinID>`
REQUIRED. Indicates the session ID which is assigned by the peer device. If the control point manages the session, it should know the session id of both sides. The control point which created the session knows the information about the session but a 3rd party control point does not know this information. So the control point which is newly added in network can manage the input session using this information.

Note: Additional elements or attributes may also be present, for example, defined by individual vendors or future versions of the *InputConfig* service specification. Consequently, a Control Point must gracefully ignore any additional elements or attributes that it does not understand.

2.4.4 *RequiredInputType*

This state variable represents the required input mode for the user interaction. When the required input mode of the receiver is changed, then the receiver device notifies this change to the interested ICP.

For example, if a user is able to use his mobile phone as an input device (sender) which supports multiple input modes such as text and coordinates to use with a web browser. The required input for the browser application may change many times in a short interval. In this case, if the receiver's information about the current input mode is delivered to the sender, it is very useful to construct or modify the UI of the sender.

This state variable can have the same value of the inputMode element in the *DeviceInputCapability* state variable.

2.4.4.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *RequiredInputType* in the XML namespace "`urn:schemas-upnp-org:phone:ics`" which is located at "`http://www.upnp.org/schemas/phone/ics.xsd`".

2.4.4.2 Description of fields in the *RequiredInputType* structure

```
<?xml version="1.0" encoding="utf-8"?>
<ics:requiredInputType
   xmlns:ics="urn:schemas-upnp-org:phone:ics"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:phone:ics
   http://www.upnp.org/schemas/phone/ics.xsd">
   <inputMode>
      <inputDeviceType>simple type</inputDeviceType>
      <inputDeviceSubClass>USB device class</inputDeviceSubClass>
      <inputDataformat type="USB_RD">
         <reportDescriptor>
              USB report descriptor
         </reportDescriptor>
      </inputDataformat>
      <transportProtocol>BT_HID_REQUEST</transportProtocol>
      <supportSecureChannel>
         <TLS>True or False</TLS>
         <DTLS>True or False</DTLS>
      </supportSecureChannel>
   </inputMode>
   <currentPossibleSessionID>sessionID</currentPossibleSessionID>
</ics:requiredInputType>
```

`<xml>`
REQUIRED. Case sensitive.

`<requiredInputType>`
REQUIRED. MUST include a namespace declaration for the InputConfig service requiredInputType Schema (“urn:schemas-upnp-org:phone:ics”). MUST include at least one of the following elements. This namespace defines the following elements and attributes:

`<inputMode>`
REQUIRED. Indicates a set of parameters supported by the input device.

`<inputDeviceType>`
REQUIRED. Indicates the simple type of the input device. This element can have a value such as Keyboard, Keypad, Remocon, Mouse etc.

`<inputDeviceSubClass>`
OPTIONAL. Indicates device subclass defined in USB HID specification. If an input device supports the USB data format this element give information to the receiver’s USB device driver.

`<inputDataformat>`
REQUIRED. Indicates a type and format of the input data to make a consensus between sender and receiver.

`type`
REQUIRED. xsd:string, Indicates the type of data format for the input data. The meaning of the format is the encoding type or packetizing type of the input data. Basically the type should have a USB_RD as a default value. If vendor want to adopt new type of input data format, the value of this element can be defined by vendor.

`<reportDescriptor>`
REQUIRED. Indicates the report descriptor of the input data only if the value of the inputDataformat@type is USB_RD. The report descriptor should comply with the USB report descriptor standard.

`<transportProtocol>`
REQUIRED. Indicates a feasible transport protocol for the input data. Basically this element should have a BT_HID_REQUEST as a default value. If vendor want to adopt a new transport protocol, the value of this element can be defined by the vendor.

`<supportSecureChannel>`
OPTIONAL. Indicates whether the input device support a method to protect input data stream.

`<TLS>`
REQUIRED. Indicates whether the input device support TCP channel protection. This element should have TRUE or FALSE as a value of the element.

`<DTLS>`
REQUIRED. Indicates whether the input device support UDP channel protection.

This element should have TRUE or FALSE as a value of the element.

```
<currentPossibleSessionID>
```
OPTIONAL. Indicates existing session id which is compliant with the required input mode.

Note: Additional elements or attributes may also be present, for example, defined by individual vendors or future versions of the InputConfig service specification. Consequently, a Control Point must gracefully ignore any additional elements or attributes that it does not understand.

2.4.5 *A_ARG_TYPE_ConnectionInfo*

This state variable is introduced to provide type information for the *ConnectionInfo* argument in the *SetInputSessino()* action. It contains information for the sender to connect to the receiver using a specific connection protocol.

However, the *ConnectionInfo* from the sender device does not have any useful information. It is just used for the Control Point to receive symmetric output argument in the *SetInputSessino()* action

2.4.5.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *A_ARG_ConnectionInfo* in the XML namespace “`urn:schemas-upnp-org:phone:ics`” which is located at “`http://www.upnp.org/schemas/phone/ics.xsd`”.

2.4.5.2 Description of fields in the *A_ARG_TYPE_ConnectionInfo* structure

```
<?xml version="1.0" encoding="utf-8"?>
<ics:connectionInfo
   xmlns:ics="urn:schemas-upnp-org:phone:ics"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:phone:ics
   http://www.upnp.org/schemas/phone/ics.xsd">
   <sessionParameters>
      <deviceRole>role of the device</deviceRole>
      <RDtransportPortNum>
         port number used for Report descriptor transport
      </RDtransportPortNum>
      <controlPortNum>
         port number used for control channel
      </controlPortNum>
      <interruptPortNum>
         port number used for interrupt channel
      </interruptPortNum>
      <secureControlPortNum>
         TLS port number used for control secure channel
      </secureControlPortNum>
      <secureInterruptPortNum>
         DTLS port number used for interrupt secure channel
      </secureInterruptPortNum>
         Additional element description for user defined connectionInfo
   </sessionParameters>
</ics:connectionInfo>
```

`<xml>`REQUIRED. Case sensitive.

`<connectionInfo>`
REQUIRED. MUST include a namespace declaration for the InputConfig service connectionInfo Schema ("urn:schemas-upnp-org:phone:ics"). MUST include one or more of the following elements. This namespace defines the following elements and attributes:

`<deviceRole>`
REQUIRED. Indicates the role of the target input device, so that Control Point can know the device's role and decide whether this information will be used for connection or to be discarded.

`<RDtransportPortNum>`
REQUIRED. Indicates the port number to transfer the report descriptor.

`<controlPortNum>`
REQUIRED. Indicates the port number to be used for control channel for input data session.

`<interruptPortNum>`
REQUIRED. Indicates the DTLS port number to be used for interrupt channel for input data session.

`<secureControlPortNum>`
REQUIRED. Indicates the TLS port number to be used for secure control channel for input data session.

`<secureInterruptPortNum>`
REQUIRED. Indicates the port number to be used for secure interrupt channel for input data session.

Note: Additional elements or attributes may also be present, for example, defined by individual vendors or future versions of the InputConfig service specification. Consequently, a Control Point must gracefully ignore any additional elements or attributes that it does not understand.

2.4.6 *A_ARG_TYPE_InputCapability*

This state variable is introduced to provide type information for the *SelectedCapability* argument in the *SetInputSession()* and *StartInputSession()* action.

This state variable contains the selected set of input capabilities for making an input session. This *A_ARG_TYPE_ InputCapability* state variable contains the value of the selected role, input mode, transport protocols, and input data format that the Control Point has selected.

2.4.6.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *A_ARG_TYPE_ InputCapability* in the XML namespace "`urn:schemas-upnp-org:phone:ics`" which is located at "`http://www.upnp.org/schemas/phone/ics.xsd`".

2.4.6.2 Description of fields in the *A ARG TYPE InputCapability* structure

```
<?xml version="1.0" encoding="utf-8"?>
<ics:inputCapability
   xmlns:ics="urn:schemas-upnp-org:phone:ics"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:phone:ics
   http://www.upnp.org/schemas/phone/ics.xsd">
   <capability role="sender">
      <inputMode>
         <inputDeviceType>text</inputDeviceType>
         <inputDeviceSubClass>USB device class</inputDeviceSubClass>
         <inputDataformat type="USB_RD">
            <reportDescriptor>
               USB report descriptor
            </reportDescriptor>
         </inputDataformat>
         <transportProtocol>BT_HID_REQUEST</transportProtocol>
         <supportSecureChannel>
```

```
                <TLS>True or False</TLS>
                <DTLS>True or False</DTLS>
            </supportSecureChannel>
        </inputMode>
    </capability>
</ics:inputCapability>
```

`<xml>`
REQUIRED. Case sensitive.

`<inputCapability>`
REQUIRED. MUST include a namespace declaration for the InputConfig service InputCapability Schema (“urn:schemas-upnp-org:phone:ics”). MUST include one or more of the following elements. This namespace defines the following elements and attributes:

`<capability>`
REQUIRED. Indicates required parameters to make input sessions.

`Role`
REQUIRED. Xsd:string, Indicates the roles Input device is MUST be in charge of. For example, sender means that this Input device can send the input data to its pair.

`<inputMode>`
REQUIRED. Indicate a set of parameters supported by input device.

`<inputDeviceType>`
REQUIRED. Indicate simple type of the input device. This element can have a value such as Keyboard, Keypad, Remocon, Mouse.

`<inputDeviceSubClass>`
OPTIONAL. Indicate device subclass defined in USB HID specification. If input device support the USB data format this element give information to the receiver's USB device driver.

`<inputDataformat>`
REQUIRED. Indicates a type and format of the input data to make a consensus between sender and receiver.

`type`
REQUIRED. xsd:string, Indicates the type of data format for the input data. The meaning of the format is the encoding type or packetizing type of the input data. Basically the type should have a USB_RD as a default value. If vendor want to adopt new type of input data format, the value of this element can be defined by vendor.

`<reportDescriptor>`
REQUIRED. Indicates the report descriptor of the input data only if the value of the inputDataformat@type is USB_RD. The report descriptor should comply with the USB report descriptor standard.

`<transportProtocol>`
REQUIRED. Indicates a feasible transport protocol for the input data. Basically this element should have a BT_HID_REQUEST as a default value. If vendor want to adopt a new transport protocol, the value of this element can be defined by the vendor.

`<supportSecureChannel>`
OPTIONAL. Indicates whether the input device support a method to protect input data stream.

`<TLS>`
REQUIRED. Indicates whether the input device support TCP channel protection. This element should have TRUE or FALSE as a value of the element.

`<DTLS>`
REQUIRED. Indicates whether the input device support UDP channel protection. This element should have TRUE or FALSE as a value of the element.

Note: Additional elements or attributes may also be present, for example, defined by individual vendors or future versions of the *InputConfig* service specification. Consequently, a Control Point must gracefully ignore any additional elements or attributes that it does not understand.

2.4.7 A_ARG_TYPE_DeviceInfo

This state variable is introduced to provide type information for the *PeerDeviceInfo* argument in the *SetInputSession()* action.

Before a control point discovers the input devices, it identifies the UUIDs of the devices. The Control Point can configure the PeerDeviceUUID of the sender and the receiver through these UUIDs.

Each input devices should update its current session information *InputConnectionList* using the *PeerDeviceInfo* information.

This *A_ARG_TYPE_DeviceInfo* includes deviceCertID element that represents the device identity of input devices, which is defined in DeviceProtecion section 2.2.2.1. The identity is exchanged between sender and receiver when the ICP configures *PeerDeviceInfo* of the input devices. While the input devices handshakes TLS parameters, the identities are also exchanged and used to authenticate each others.

2.4.7.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *A_ARG_TYPE_DeviceInfo* in the XML namespace "`urn:schemas-upnp-org:phone:ics`" which is located at "`http://www.upnp.org/schemas/phone/ics.xsd`".

2.4.7.2 Description of fields in the *A_ARG_TYPE_DeviceInfo* structure

```
<?xml version="1.0" encoding="utf-8"?>
<ics:deviceInfo
   xmlns:ics="urn:schemas-upnp-org:phone:ics"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:phone:ics
   http://www.upnp.org/schemas/phone/ics.xsd">
   <deviceFriendlyName>Omnia</deviceFriendlyName>
   <deviceUUID>6E09886B-DC6E-439F-82D1-7CCAC7F4E3B4</deviceUUID>
   <deviceCertID>45A27C62-1EEC-2E23-F32E-ABC22EF1E2BE</deviceCertID>
   <peerSessionID>2</peerSessionID>
</ics:deviceInfo>
```

`<xml>`
REQUIRED. Case sensitive.

`<deviceInfo>`
REQUIRED. MUST include a namespace declaration for the InputConfig service PeerDeviceInfo Schema (“urn:schemas-upnp-org:phone:ics”). MUST include one or more of the following elements. This namespace defines the following elements and attributes:

`<deviceFriendlyName>`
REQUIRED. Indicates the Friendly name of the peer device.

`<deviceUUID>`
REQUIRED. Indicates the UUID of the peer device which is used by a 3rd party Control Point to discover peer device.

`<deviceCertID>`
OPTIONAL. Indicates the device identity for authentication and used to create a secure data and control channel between a sender and the receiver. The value ofdevice’s identity is defined in DeviceProtecion section 2.2.2.1.

`<peerSessionID>`
OPTIONAL. Indicates the sessionID of the peer device used for 3rd party Control Point to discover peer device and terminate or modify an existing session.

Note: Additional elements or attributes may also be present, for example, defined by individual vendors or future versions of the *InputConfig* service specification. Consequently, a Control Point must gracefully ignore any additional elements or attributes that it does not understand.

2.4.8 *A_ARG_TYPE_InputSessionID*

This state variable is introduced to provide type information for the *SessionID* argument in the *SetInputSession()*, *StartInputSession()*, and *StopInputsession()* actions.

The Control Point identifies an input session by the *SessionID*, and uses it to setup a new session or to terminate an existing session. The Input devices keep its list of active sessions' *SessionID* in the form of XML *InputConnectionList*. If the device receives the peer's *sessionID* from the Control Point or generates it by itself then the device should add the *SessionID* to the *InputConnectionList*.

2.4.9 *A_ARG_TYPE_MultiInputMode*

This state variable is introduced to provide type information for the *MultiInputMode* argument in the *SetMultiInputMode()* action.

A Control Point is able to set the value of the multiInputMode element in the *InputConnectionList* state variable and also change it through the *SetMultiInputMode()* action. The Control point selects the input mode of receiver, which decides the rule, and how the receiver handles the two or more input data streams from senders of a lot.

Only the receiver may support multi input mode and the *SetMultiInputMode()* action.

Multi input mode defines a set of rules to deal with input data from senders of a lot. There are three types of modes defined in this version of the InputConfig service.

Mode #1(monopolized mode): Only one sender has the ownership to handle the receiver at a time. The other sender can't access the receiver if the first sender does not release the ownership.

Mode #2(default mode): All sender access the receiver and receiver manages the multiple inputs one by one. (Similar to the current operation of USB HID)

Mode #3(independent mode): A receiver classifies and independently processes the input data according to its source. InputConfig service makes different sessions for each source of input data and allocates the session ids to the each session. Even though the input data is transferred from same source, the receiver device distinguishes the sources of the data using session id.(Similar to the current operation of Joystick)

2.5 Eventing and Moderation

Table 2-3: Eventing and Moderation

Variable Name	Evented	Moderated[1]	Criteria
DeviceInputCapability	*YES*	*NO*	
RequiredInputType	*YES*	*NO*	

[1] *YES* = The state variable MUST be moderated with the criteria

2.5.1 *DeviceInputCapability*

This state variable sends event when the capability supported by input device is change. For example, if the data channel is currently protected using DTLS and the device decides not to use any protection mechanism anymore, the input device may disable the DTLS function and change the corresponding entry in the *DeviceInputCapability*. The device then notifies this change of the *DeviceInputCapability* to the interested

ICPs. Therefore the *InputConfig* service MUST monitor whether the capability of the input device (receiver) has been changed.

2.5.2 *RequiredInputType*

This state variable sends events for the receiver device to notify the change to the interested ICP when the required input mode of the receiver is changed,

2.6 Actions

Table 2-4: Actions

Name	Device R/O[1]	Control Point R/O[2]
GetInputCapability()	R	R
GetInputConnectionList()	R	R
SetInputSession()	R	R
StartInputSession()	R	R
StopInputSession()	R	R
SwitchInputSession()	R	R
SetMultiinputMode()	R	R
SetMonopolizedSender()	R	R

[1] For a device this column indicates whether the action MUST be implemented or not, where *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*)..

[2] For a control pont this column indicates whether a Control Point MUST be capable of invoking this action, where *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*)..

2.6.1 *GetInputCapability()*

This action returns information of the input device's capability such as input device type, transport protocol and so on. The state variable *DeviceInputCapability* includes this information in an xml fragment.

.

2.6.1.1 Arguments

Table 2-5: Arguments for *GetInputCapability()*

Argument	Direction	relatedStateVariable
SupportedCapabilities	*OUT*	*DeviceInputCapability*

2.6.1.2 Argument Descriptions

This argument follows the format of the *DeviceInputCapability* state variable

2.6.1.3 Service Requirements

This service MUST check the current status of the device resources and reflect resources' status in the state variable. The transport protocol stack for image input data may not be supported for some reasons, or the remaining resources may be too small to handle real-time mouse input datum etc. and hence the *DeviceInputCapability* should be modified according to the resource condition of the device.

2.6.1.4 Control Point Requirements When Calling the Action

The Control Point MUST keep this information to setup a new session. When the Control Point adds a new session, it can use this information rather than invoking the same action again. The Control Point MUST keep device's resource status and update the status based on the event of the *DeviceInputCapability* state variable.

2.6.1.5 Dependency on Device State

None.

2.6.1.6 Effect on Device State

None.

2.6.1.7 Errors

Table 2-6: Error Codes for *GetInputCapability()*

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.

2.6.2 *GetInputConnectionList()*

This action returns information about all existing input sessions including their status and the devices that are connected by each input session. The state variable *InputConnectionList* includes this information in an xml fragment.

If the Control Point has the capability to keep information related to the sender and the receiver which are connected for an input session by this Control Point, then the control point can disconnect the existing session or modify the existing session by invoking the *SetInputSession()* action. However, it can't be assumed that all the Control Points are able to store session related information and also a 3rd party Control Point is not able to recognize an existing session without invoking this action

2.6.2.1 Arguments

Table 2-7: Arguments for *GetInputConnectionList()*

Argument	Direction	relatedStateVariable
CurrentConnectionList	*OUT*	*InputConnectionList*

2.6.2.2 Argument Descriptions

This argument follows a format of the *InputConnectionList*.

2.6.2.3 Service Requirements

The device whose role is a receiver MUST manage input connection but another input service whose role is a sender doesn't need to manage it.

2.6.2.4 Control Point Requirements When Calling the Action

None.

2.6.2.5 Dependency on Device State

None.

2.6.2.6 Effect on Device State

None.

2.6.2.7 Errors

Table 2-8: Error Codes for *GetInputConnectionList()*

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.

2.6.3 *SetInputSession()*

This action configures the input capability information for an input session. The state variable *SelectedCapability* is selected from a list of capabilities defined in the state variable *DeviceInputCapability*. The configured capability information includes the role of the device and the other information which are synced between the sender and the receiver. After receiving this action, the input device sets their role as the sender or the receiver.

The action returns the *SessionID* for the input session which is unique within an input device, and the *ConnectionInfo* which is dynamically generated by the device to create the connection. The type of information included in the *ConnectionInfo* depends on the transport protocol supported by the receiver.

To allow a third party control point to control an input session, the *InputConnectionList* includes peer device's information in the *PeerDeviceInfo* state variable. The *PeerDeviceInfo* state variable includes PeerDeviceUUID and PeerSessionID. The *PeerDeviceInfo* input argument includes values for both sender and receiver. But the PeerSessionID of the *PeerDeviceInfo* does not include a value when this action is invoked to configure the receiver first because the *SessionID* of the sender is generated only when the action is invoked on the sender.

When the control point configures a device with the receiver role by invoking the action *SetInputSession()*, the value of the input argument *PeerDeviceInfo* contains an empty string ("") since the *PeerDeviceInfo* is not required to configure the receiver. The *PeerDeviceInfo* is only required to configure the sender. The control point MUST configure the receiver first before configuring the sender.

2.6.3.1 Arguments

Table 2-9: Arguments for *SetInputSession()*

Argument	Direction	relatedStateVariable
SelectedCapability	*IN*	*A_ARG_TYPE_InputCapability*
Receiverinfo	*IN*	*A_ARG_TYPE_ConnectionInfo*

Argument	Direction	relatedStateVariable
PeerDeviceInfo	*IN*	*A_ARG_TYPE_DeviceInfo*
SessionID	*OUT*	*A_ARG_TYPE_InputSessionID*
ConnectionInfo	*OUT*	*A_ARG_TYPE_ConnectionInfo*

2.6.3.2 Argument Descriptions

The input argument *SelectedCapability* contains the input capability of the input device, which is selected by ICP invoking this action.

The input argument *Receiverinfo* contains the connection information of receiver device when the ICP invokes this action to sender device for configuration.

The input argument *PeerDeviceInfo* contains the identity of the peer device which will be connected to the input device that is configuring by ICP.

The output argument *SessionID* contains the session id of the session made by ICP which is invoking this action.

The output argument *ConnectionInfo* contains connection information of receiver device when the ICP invokes this action to receiver device to for configuration.

2.6.3.3 Service Requirements

This service MUST manage the input session and it is identified the *SessionID*. Using this ID, ICP start and stop the input data transfer through the input session.

2.6.3.4 Control Point Requirements When Calling the Action

The control point MUST select the input capability of the sender and the receiver for the state variable *SelectedCapability.* The control point MUST also maintain the *SessionID*s for the sender and the receiver for the input session. The control point may later use the *SessionID*s to delete the input session from the sender and the receiver side.

The Control Point MUST first invokes this action on the receiver and then invokes the same action on the sender. The receiver's connection information is transferred to the sender in the *ConnectionInfo* state variable. .

2.6.3.5 Dependency on Device State

None.

2.6.3.6 Effect on Device State

None.

2.6.3.7 Errors

Table 2-10: Error Codes for *SetInputSession()*

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	Invalid SelectedCapability	SelectedCapability is invalid

ErrorCode	errorDescription	Description
702	Invalid Receiverinfo	Receiverinfo is invalid
703	Invalid PeerDeviceInfo	PeerDeviceInfo is invalid

2.6.4 *StartInputSession()*

This action initiates the input session for data transfer. The input argument *SessionID* identifies the input session to be started for data transfer. This action will fail with an error code when the *SessionID* does not exist.

2.6.4.1 Arguments

Table 2-11: Arguments for *StartInputSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_InputSessionID*

2.6.4.2 Argument Descriptions

The input argument *SessionID* is the unique ID of the input connection.

2.6.4.3 Service Requirements

This service MUST manage the input session and it is identified the *SessionID*.

2.6.4.4 Control Point Requirements When Calling the Action

The Control Point MUST invoke this action according in an order- the control point first invoke this action on the receiver and then on the sender. The receiver first prepares itself for receiving the input data from the sender. The *ConnectionInfo* that is generated by the receiver is delivered to the sender. Dependency on Device State

2.6.4.5 Effect on Device State

None.

2.6.4.6 Errors

Table 2-12: Error Codes for *StartInputSession()*

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.
704	Invalid SessionID	The Session ID is invalid when the session does not exist.

2.6.5 *StopInputSession()*

This action terminates an existing input session which is identified by input argument sessionID. This action will fail with an error code if the *SessionID* does not exist or the *SessionID* is invalid.

2.6.5.1 Arguments

Table 2-13: Arguments for *StopInputSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_InputSessionID*

2.6.5.2 Argument Descriptions

The input argument *SessionID* is the unique ID of the input connection.

2.6.5.3 Service Requirements

This service MUST manage the input session and it is identified the *SessionID*.

2.6.5.4 Control Point Requirements When Calling the Action

Control Points should terminate the session of both sender and receiver explicitly.

2.6.5.5 Dependency on Device State

None.

2.6.5.6 Effect on Device State

None.

2.6.5.7 Errors

Table 2-14: Error Codes for *StopInputSession()*

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.
704	Invalid SessionID	The Session ID is invalid when the session does not exist.

2.6.6 *SwitchInputSession()*

This action switches the currently used input session to a different one. The input argument *SessionID* identifies the new input session the sender and the receiver will use for the transfer of input data.

The receiver in an existing input session may change the type of input it requires and notifies the new input type by an event to the control point. The control point may choose an existing input session with the same input type by invocation of this action.

2.6.6.1 Arguments

Table 2-15: Arguments for *SwitchInputSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_InputSessionID*

2.6.6.2 Argument Descriptions

The input argument *SessionID* is the unique ID of the input connection.

2.6.6.3 Service Requirements

This service MUST manage the input session and it is identified the *SessionID*.

2.6.6.4 Control Point Requirements When Calling the Action

The Control Point MUST invoke this action when it receives an event about the change of input mode from a receiver device. If there is an existing input session with the new session mode already exist between the sender and the receiver, then the control point MUST invoke this action at the sender to switch the input session to the new session identified by the SessionID input argument. The event message sent by the receiver to the control point includes the session id of the new input mode and the session already exist etween the receiver and the sender. If no session exists between the receiver and the sender with the new input mode then the control point MUST create a new session for the input mode. The value of "0" in the sessionID in the vent states that a session for the new input mode does not exist.

2.6.6.5 Dependency on Device State

None.

2.6.6.6 Effect on Device State

None.

2.6.6.7 Errors

Table 2-16: Error Codes for *SwitchInputSession()*

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.
704	Invalid Session ID	The Session ID is invalid when the session does not exist.

2.6.7 *SetMultiinputMode()*

This action changes the current way to manage inputs from senders of a lot. The input argument *NewMultiInputMode* identifies the current mode of operation for dealing with multiple senders. The detailed description for different types of input mode to deal with inputs from multiple senders is described in section 2.4.9.

2.6.7.1 Arguments

Table 2-17: Arguments for *SetMultiinputMode()*

Argument	Direction	relatedStateVariable
NewMultiInputMode	*IN*	*A_ARG_TYPE_MultiInputMode*

2.6.7.2 Argument Descriptions

The input argument *NewMultiInputMode* is the unique ID of the input connection.

2.6.7.3 Service Requirements

This service MUST manage the rule, how the receiver processes multiple inputs from senders. The rule is selected by ICP which invokes this action.

2.6.7.4 Control Point Requirements When Calling the Action

None.

2.6.7.5 Dependency on Device State

None.

2.6.7.6 Effect on Device State

None.

2.6.7.7 Errors

Table 2-18: Error Codes for *SetMultiinputMode()*

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.
705	Invalid NewMultiInputMode	NewMultiInputMode is invalid

2.6.8 *SetMonopolizedSender()*

This action gives an ownership to a designated sender when the multiInputMode element in the *InputConnectionList* of the receiver is set to the Mode#1(monopolized mode).

The Control Point invokes the *setMonopolizedSender()* action on the receiver with the device information of the sender and the session id of the receiver. By doing so, Receiver can know which sender have its ownership. If a sender gets an ownership, other devices can’t access the receiver even though the connection is already established.

2.6.8.1 Arguments

Table 2-19: Arguments for *SetMonopolizedSender()*

Argument	Direction	relatedStateVariable
OwnerDeviceInfo	*IN*	*A_ARG_TYPE_DeviceInfo*
OwnedSessionID	*IN*	*A_ARG_TYPE_InputSessionID*

2.6.8.2 Argument Descriptions

The input argument *OwnerDeviceInfo* contains the device information of the sender which preempts the receiver.

The input argument *OwnedSessionID* contains session id which will be preempted by authorized sender.

2.6.8.3 Service Requirements

This service MUST manage the list, which describe the relation between input session and session owner.

2.6.8.4 Control Point Requirements When Calling the Action

None.

2.6.8.5 Dependency on Device State

None.

2.6.8.6 Effect on Device State

None.

2.6.8.7 Errors

Table 2-20: Error Codes for *SetMonopolizedSender()*

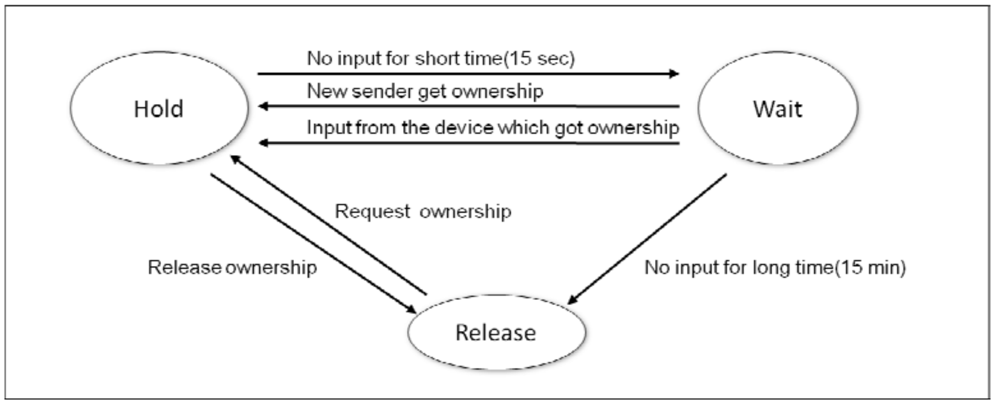
ErrorCode	errorDescription	Description
400-499	TBD	See UpnP Device Architecture section on Control.
500-599	TBD	See UpnP Device Architecture section on Control.
600-699	TBD	See UpnP Device Architecture section on Control.

2.7 Relationship between Actions

None.

2.8 Service Behavioral Mode

2.8.1 State Diagrams


Hold
Wait
Release
No input for short time(15 sec)
New sender get ownership
Input from the device which got ownership
Request ownership
Release ownership
No input for long time(15 min)

This diagram represents the state machine of a receiver which multiInputMode element in the *InputConnectionList* state variable is configured to 1.(Monopolization mode)

If receiver cannot be connected by the sender, the state of the receiver turns to the default state. (Release)

If the receiver connects to the sender, the sender receives the ownership to send input data. And the state of the receiver will be changed to Hold. In the Hold state, the receiver does not get the input from the input device that does not have the ownership.

If there is no input transmission for a short time (e.g. 15 sec) from the sender which has the ownership, the state will change to Wait. At the Wait state, if the owner again sends input to the receiver, the receiver's status turn to Hold again. If another sender gets the ownership while the receiver is in the Wait state then the state of the receiver changes to the Hold status as well.

If there is no input for a long time (e.g. 15 min) from the sender at the Wait state, the receiver moves its state to Release.

3 Theory of Operation (Informative)

3.1.1 Configuration to establish input sessions

The *InputConfig* service is embedded in input devices and the service keeps information about the capabilities of the devices. To transfer input data from the sender input device to receiver input device, the *InputConfig* service provides interfaces to configure the information related to the data transfer.

The Control Point starts the configuration by requesting the input capability of the devices such as supported input data type and the transport protocol (e.g. *GetInputCapability()*) and gets the information in the response message for the request. Based on the capabilities of the input device, the Control Point configures the input devices to setup an input session. The input client (sender) receives input data from the users and sends the data to the input server (receiver).

In the case that there are two or more input Senders, the control point performs the same process as the one used to configure each input Sender. The Control Point gets the input capabilities of the devices and configures the session between the input devices. Because the input Receiver (receiver) receives input data from multiple input Senders (sender), the input Receiver should distinguish the source of the input data and decides the rule to use the input data from multiple sources.

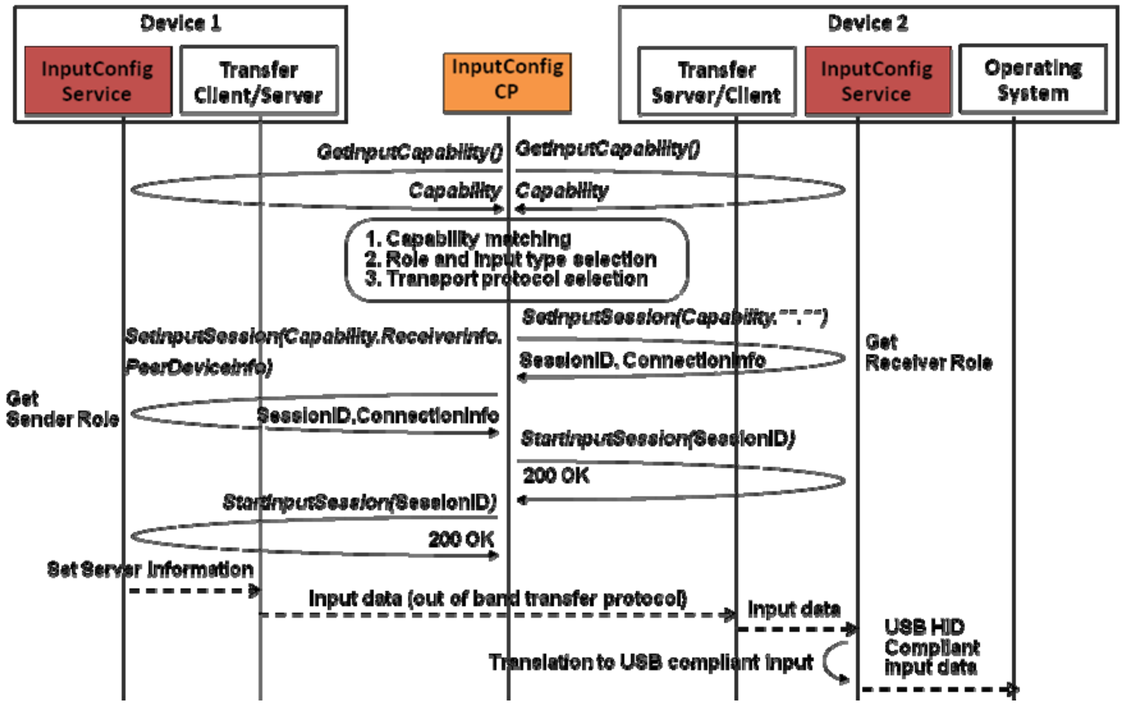
Based on the information received from each device, the Control Point can decide the configuration options such as the roles for each device and transport protocol to be used to setup the input session between a pair of devices. Among the input devices, the Control Point can select the pairs to be participated in the input session. One of the devices in the pair is a sender and the other is the receiver. The pair should have at least one input type and transport protocol supported by both of the devices.

When the matching procedure between the devices is successfully completed, the Control Point selects an input device as a receiver and invokes the *SetInputSession()* first. The input device sets its role and prepares for the connection as a receiver. It opens a port and waits for a request from the sender to receive the data. Once the preparation is completed, the device adds the information (e.g. IP address, port number) to the *ReceiverInfo* argument for the sender to connect to the receiver. The receiver returns its own *sessionID* and *ConnectionInfo* to the Control Point.

If the configuration process is over, the sender is required to configure with more parameters than the receiver did. That is the reason why the control point first configures the receiver. The Control Point invokes the *SetInputSession()* with *ReceiverInfo* and *PeerDeviceInfo* to the sender. The *ReceiverInfo* includes the IP address and port number for sender to connect to the receiver. And the *PeerDeviceInfo* includes the uuid and session id of the peer device. This information is added to the *InputConnectionList* and is used for 3rd party control point to terminate both session of receiver and sender.

After receiving this information, the sender configures the related parameters and arranges the input connection. The sender also returns the sender's session id and sender's *ConnectionInfo*. This *ConnectionInfo* from the sender is a dummy output argument for the *SetInputSession()* and is used just for representing the role of the device. At this moment, both devices finish setting their parameters for the connection but the actual connection between the sender and the receiver is yet to be established.

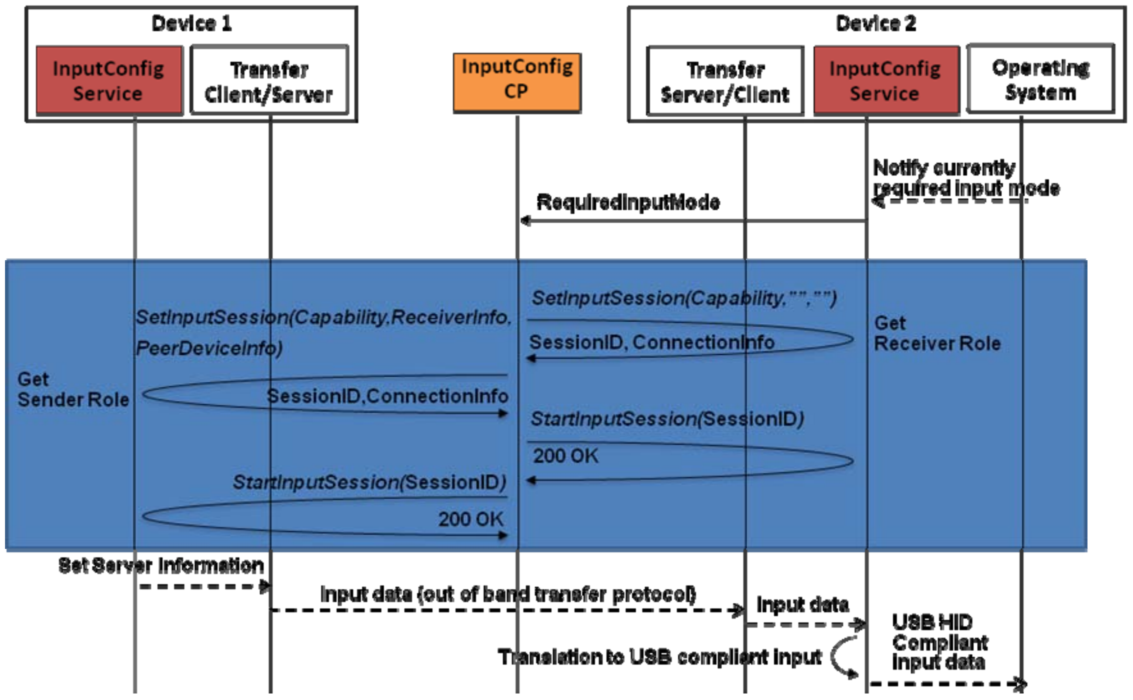
The Control Point finishes the establishment of the session through activating the session (e.g. *StartInputSession()*). The session is identified by the *sessionID*. When the Control Point finishes the configuration of the input devices, the user is able to input the data using sender device. The format of the input data also can be determined through the configuration step. The transferred input data is regarded as a native input by the receiver device.



Device 1
InputConfig Service
Transfer Client/Server
InputConfig CP
Device 2
Transfer Server/Client
InputConfig Service
Operating System
GetInputCapability()
GetInputCapability()
Capability
Capability
1. Capability matching
2. Role and Input type selection
3. Transport protocol selection
SetInputSession(Capability,"","")
SetInputSession(Capability,ReceiverInfo, PeerDeviceInfo)
SessionID, ConnectionInfo
Get Receiver Role
Get Sender Role
SessionID,ConnectionInfo
StartInputSession(SessionID)
200 OK
StartInputSession(SessionID)
200 OK
Set Server Information
Input data (out of band transfer protocol)
Input data
USB HID Compliant input data
Translation to USB compliant input

Figure 3-1: Input session Configuration.

The input type that the receiver would like to receive can be changed during the operation of the input devices. In this case, the receiver sends an event notifying a change of the required input type. From this notification, the Control Point can decide whether to setup a new session for this input type. If input type is changed to text type and there is no existing session for text input, the Control Point makes a new input session for the text input. Figure 3-2 describes the process to setup a new session for this case.



Device 1
InputConfig Service
Transfer Client/Server
InputConfig CP
Device 2
Transfer Server/Client
InputConfig Service
Operating System
Notify currently required input mode
RequiredInputMode
SetInputSession(Capability,"","")
SetInputSession(Capability,ReceiverInfo, PeerDeviceInfo)
SessionID, ConnectionInfo
Get Receiver Role
Get Sender Role
SessionID,ConnectionInfo
StartInputSession(SessionID)
200 OK
StartInputSession(SessionID)
200 OK
Set Server Information
Input data (out of band transfer protocol)
Input data
USB HID Compliant Input data
Translation to USB compliant input

Figure 3-2: Input mode change.(New session)

Although the input type is changed, if there is already an existing session for the input type, the Control Point doesn't need to make a new input session. It just switches the currently used input session by invoking an action (*SwitchInputSession()*). Then the existing session becomes active and the user can use this input function. If the sender device just provides the UI for the currently used input type, it is possible for Control Point to notify the currently required input type to the sender device to change the UI.

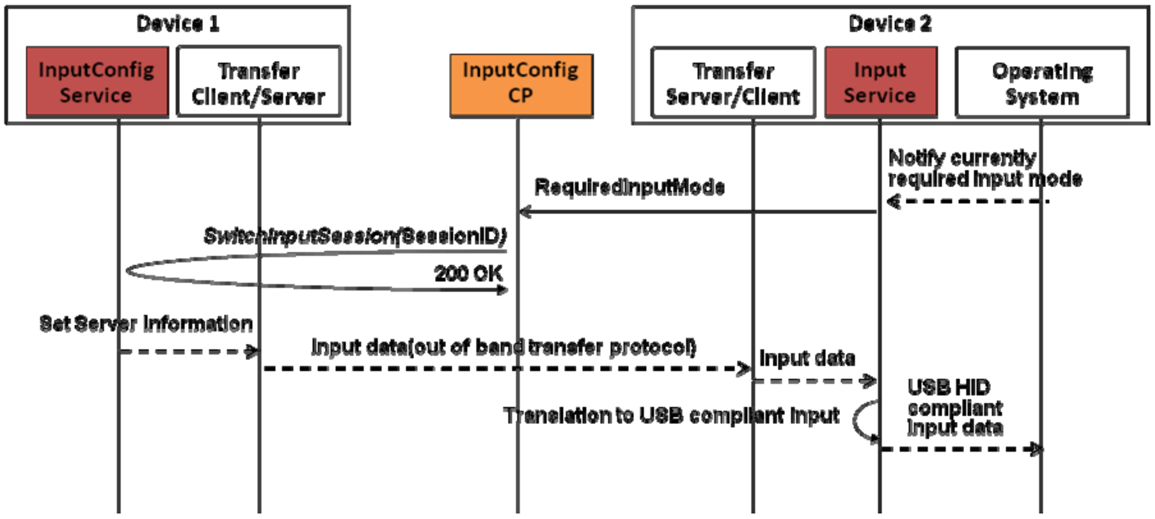


Figure 3-3*: Input mode change. (Existing Session)*

The Control Point should gracefully terminate both sessions of the input devices.

Considering the failure of the termination process and unused session for a long time, it is allowed for receiver device to finish the session when input data does not arrive for a specified amount of time.

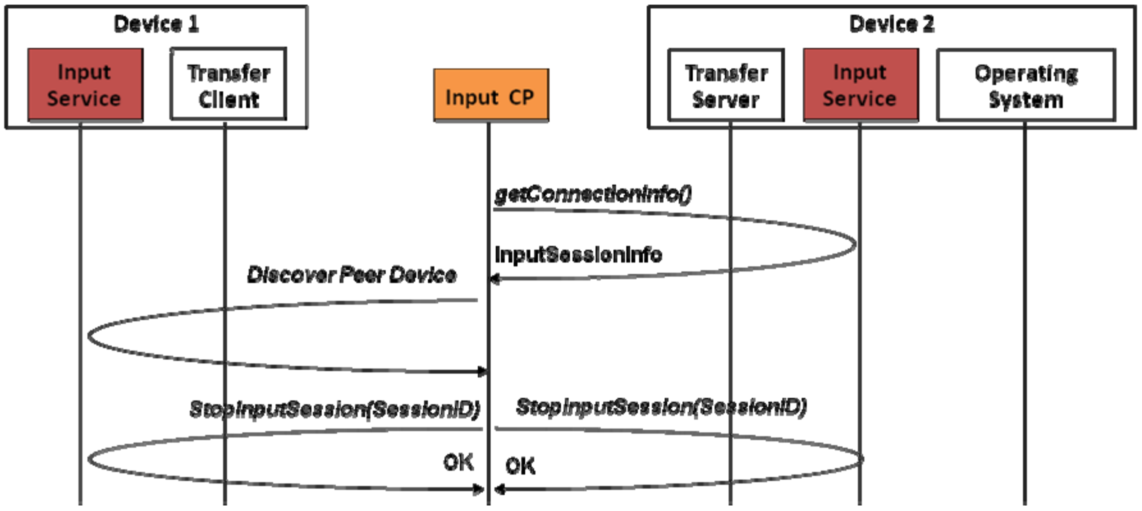


Figure 3-4: Session Termination.

3.1.2 Configuration for multiple input devices

To configure multiple devices, the InputConfig Control Point performs the same operation which it does for a single device. The Control Point distinguishes multiple input sessions by the sessionIDs. Before the Control Point sets the *multiInputMode* element in the *InputConnectionList* of the Receiver, the Receiver configures the value of the *multiInputMode* to default mode. The Control Point gets the value of the *multiInputMode* by checking the *InputConnectionList* using the *GetInputConnectionList()*. And then the Control Point set the *multiInputMode* through the *SetMultiInputMode()* action with *A_ARG_TYPE_MultiInputMode* as the argument.

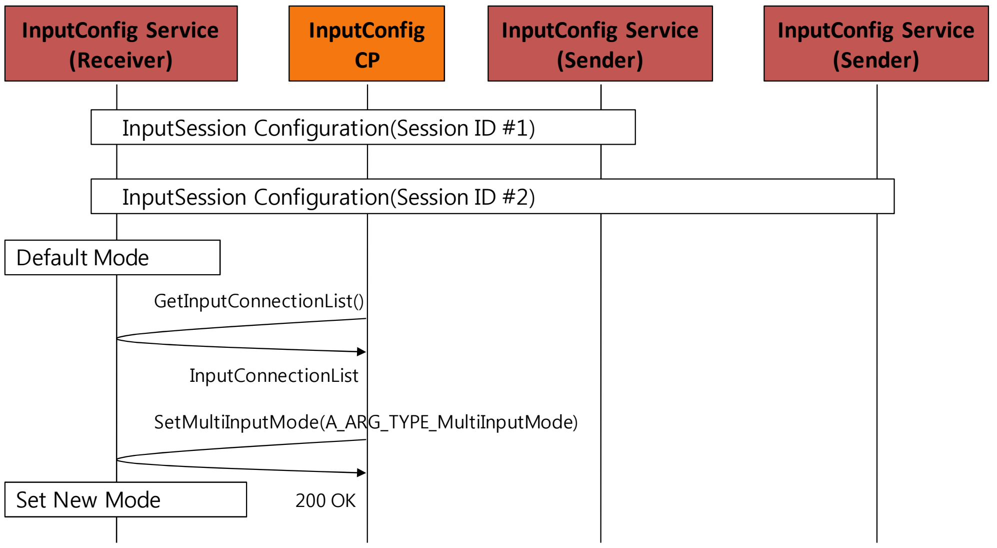

InputConfig Service (Receiver)
InputConfig CP
InputConfig Service (Sender)
InputConfig Service (Sender)
InputSession Configuration(Session ID #1)
InputSession Configuration(Session ID #2)
Default Mode
GetInputConnectionList()
InputConnectionList
SetMultiInputMode(A_ARG_TYPE_MultiInputMode)
Set New Mode
200 OK

Figure 3-5: Input mode change. (Existing Session)

If the value of the *multiInputMode* is #1, it means that the receiver regards one sender as a main sender and does not use the input data from other senders. In this case, using the *SetMonopoilizedSender(),* the control point decides main sender among many senders.

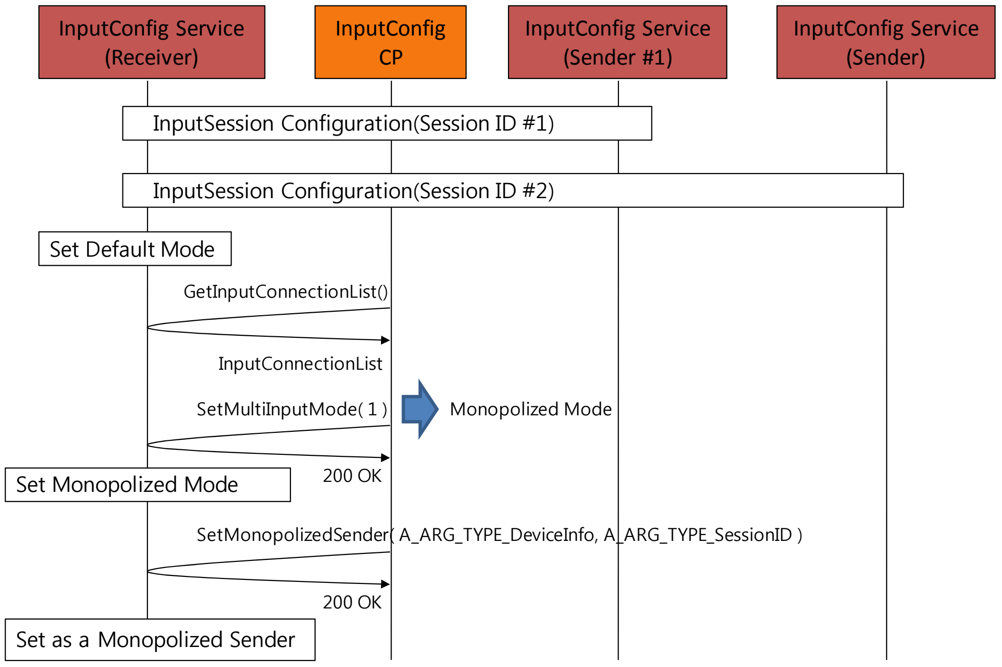

InputConfig Service (Receiver)
InputConfig CP
InputConfig Service (Sender #1)
InputConfig Service (Sender)
InputSession Configuration(Session ID #1)
InputSession Configuration(Session ID #2)
Set Default Mode
GetInputConnectionList()
InputConnectionList
SetMultiInputMode(1)
Monopolized Mode
Set Monopolized Mode
200 OK
SetMonopolizedSender(A_ARG_TYPE_DeviceInfo, A_ARG_TYPE_SessionID)
200 OK
Set as a Monopolized Sender

Figure 3-6*:* Input mode change. (Existing Session)

3.1.3 Transport protocol through TCP connection

To send input data from a sender to the receiver through the IP protocol, the input devices have to make two kinds of channels with each having different characteristics of reliability and latency.

One is a control channel through which input devices send control messages and relevant data to each other. The control channel is required to have strong reliability in order to send a control message with integrity and without any loss of data. But it does not require low latency time to send data. For this reason the InputConfig service uses the TCP channel as a control channel. The Input devices may exchange input data through control channel, but even though it is possible to do that, basically all the input data should be transferred through another channel for data except the cases where the InputConfig service needs synchronized input. The synchronized input means that the receiver requests the input from the sender, and the sender sends input data as a response of the request.

The other channel is an interrupt channel to transfer real input data such as key codes and coordinate value. Low latency time is essential feature of this channel, because input device should deliver lots of data in a short period of time. But the data integrity is not highly required for interrupt channel. Even though there are some loses of the input data, it may not be a serious problem to the user. In that case, the user may send the input data again. Therefore the InputConfig service uses the UDP channel to guarantee high speed input data transport.

These channels can be established after the process of UPnP configuration for input session such as device discovery, capability matching and session configuration. To make this channel, the input device whose role is a receiver give the *ConnectionInfo* to the other device which has the sender role. There is an element for the port information in the *ConnectionInfo* and using this information in the TCP and UDP, the Telephony clients in the sender device connect to the receiver device and setup the channel.

The InputConfig services use a protocol which is defined in Bluetooth HID profile to exchange the control signal. The protocol defines messages called REQUEST and it may contain a report, a data structure including input data.

UPnP HID adaptor described in figure 3-5 receives the control messages called REQUEST, process it and generates the USB HID signal and transmit the signal to the USB HID driver.

Based on this channel, the interaction between the input devices can be carried out. The control message through the control channel and real input data is transferred through the interrupt channel. If the receiver changes its input type, receiver sends the information about the changed input type. After receiving the input type information, the sender sends the response to the receiver with the report descriptor that the sender will use.

Basically report descriptor is included in the *DeviceInputCapability*. The state variable contains report descriptor information, but if the receiver frequently changes the required input type, the report descriptor related to the input type should be changed instantly. The InputConfig service is able to change its input type by the *RequiredInputType* and *SwitchInputSession()*. But this approach has an fatal flaw to change input type, which is that if there is no control point in a network sender cannot receive the *SwitchInputSession()*. In case of 2-box model, it is not a problem because control point and InputConfig service are in the same device, so control point is always turned on. But considering a 3-box model, if there is no control point turned on in home network, this process does not work. And even though there is a control point in the network, it takes long time to reflect the change to sender.

For this reason, the InputConfig service defined an additional REQUEST, which is named as GET_REPORT_DESCRIPTOR, to get the report descriptor. The REQUEST includes the information about the input type which is required by the receiver. The receiver sends this REQUEST to the sender and gets report descriptor from the sender when the receiver changes its required input type. If the sender does not support a report descriptor of the input type for the receiver request, the sender responds with USE_DEAULT_RD, an error code which means that the sender does not have the proper report descriptor, so it will use the default report descriptor for the input type.

If the user want to set the report descriptor on the sender side, one more additional REQUEST is needed, SET_REPORT_DESCRIPTOR. The sender sends this REQUEST to the receiver to deliver the type information and report descriptor which is supported by the sender.

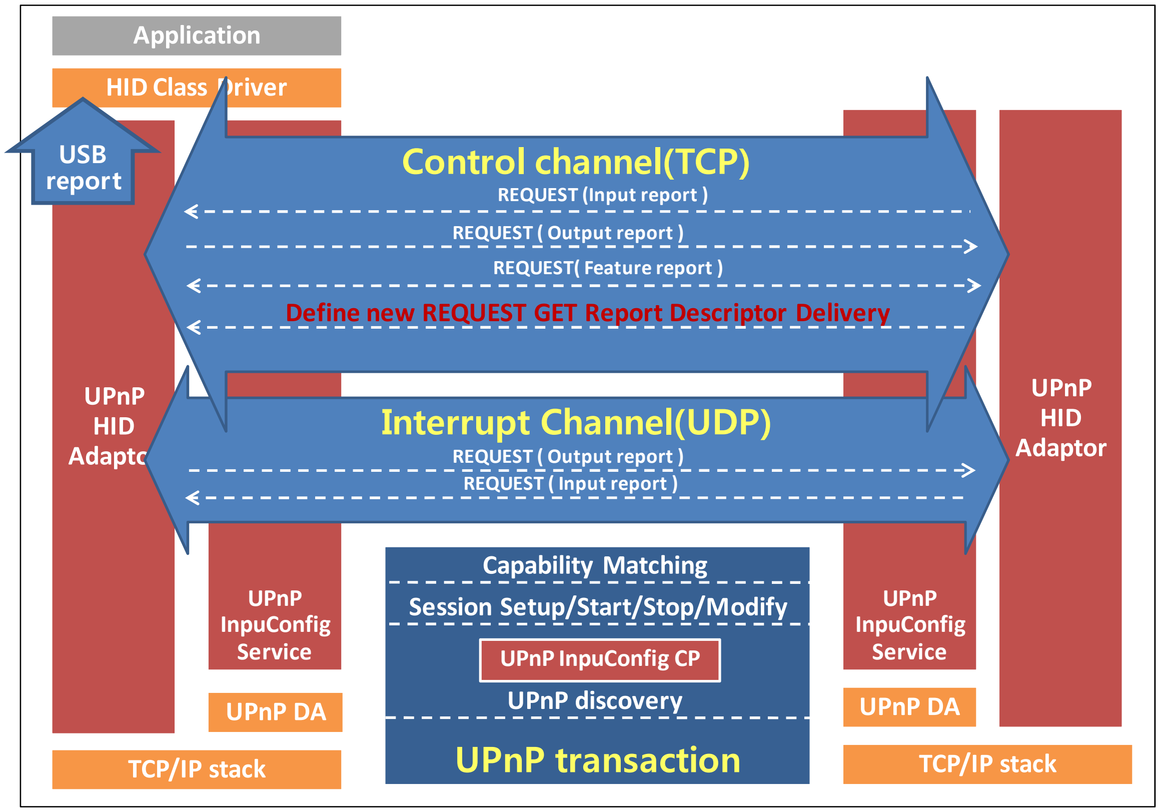


Figure 3-7: InputConfig service transport protocol diagram

Field	Size	Description
Request	1	7..4 Transaction Type 12=GET_REPORT_DESCRIPTOR
Required	1	7..6 Input Type indicator 0= USB HID Subclass 1= Vendor Specific Type 5..0 Input Type

Field	Size	Description
Request	1	7..4 Transaction Type 12=SET_REPORT_DESCRIPTOR 5..0 Reserved
Input Type		7..6 Input Type indicator 0= USB HID Subclass 1= Vendor Specific Type 5..0 Input Type

Report Descriptor	n	Report Description for Receiver

Figure 3-8: Additional REQUEST

Field	Size	Description
Request	1	7..4 Transaction Type 0=HANDSHAKErequest 3..2 ResultCode 0x0 = SUCCESSFUL …. 0x5 = USE_DEFAULT_RD ….

Figure 3-9: Additional error code

4 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>GetInputCapability</name>
         <argumentList>
            <argument>
               <name>SupportedCapabilities</name>
               <direction>out</direction>
               <relatedStateVariable>
                  DeviceInputCapability
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetInputConnectionList</name>
         <argumentList>
            <argument>
               <name>CurrentConnectionList</name>
               <direction>out</direction>
               <relatedStateVariable>
                  InputConnectionList
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetInputSession</name>
         <argumentList>
            <argument>
               <name>SelectedCapability</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InputCapability
               </relatedStateVariable>
            </argument>
            <argument>
               <name>ReceiverInfo</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_ConnectionInfo
               </relatedStateVariable>
            </argument>
            <argument>
               <name>PeerDeviceInfo</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_DeviceInfo
               </relatedStateVariable>
```

```
        </argument>
        <argument>
          <name>SessionID</name>
          <direction>out</direction>
          <relatedStateVariable>
            A_ARG_TYPE_InputSessionID
          </relatedStateVariable>
        </argument>
        <argument>
          <name>ConnectionInfo</name>
          <direction>in</direction>
          <relatedStateVariable>
            A_ARG_TYPE_ConnectionInfo
          </relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>StartInputSession</name>
      <argumentList>
        <argument>
          <name>SessionID</name>
          <direction>in</direction>
          <relatedStateVariable>
            A_ARG_TYPE_InputSessionID
          </relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>StopInputsession</name>
      <argumentList>
        <argument>
          <name>SessionID</name>
          <direction>in</direction>
          <relatedStateVariable>
            A_ARG_TYPE_InputSessionID
          </relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SwitchInputSession</name>
      <argumentList>
        <argument>
          <name>SessionID</name>
          <direction>in</direction>
          <relatedStateVariable>
            A_ARG_TYPE_InputSessionID
          </relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetMultiInputMode</name>
      <argumentList>
        <argument>
```

```
                  <name>NewMultiInputMode</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_MultiInputMode
                  </relatedStateVariable>
               </argument>
            </argumentList>
         </action>
         <action>
            <name>SetMonopolizedSender</name>
            <argumentList>
               <argument>
                  <name>OwnerDeviceInfo</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_DeviceInfo
                  </relatedStateVariable>
               </argument>
               <argument>
                  <name>OwnedSessionID</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_InputSessionID
                  </relatedStateVariable>
               </argument>
            </argumentList>
         </action>
      </actionList>
      <serviceStateTable>
         <stateVariable sendEvents="no">
            <name>DeviceInputCapability</name>
            <dataType>string</dataType>
         </stateVariable>
         <stateVariable sendEvents="no">
            <name>InputConnectionList</name>
            <dataType>string</dataType>
            </stateVariable>
         <stateVariable sendEvents="yes">
            <name>RequiredInputType</name>
            <dataType>string</dataType>
         </stateVariable>
         <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_ConnectionInfo</name>
            <dataType>string</dataType>
         </stateVariable>
         <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_InputCapability</name>
            <dataType>string</dataType>
         </stateVariable>
         <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_DeviceInfo</name>
            <dataType>string</dataType>
         </stateVariable>
         <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_InputSessionID</name>
            <dataType>ui4</dataType>
         </stateVariable>
         <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_MultiInputMode</name>
            <dataType>string</dataType>
            <defaultValue>1</defaultValue>
            <allowedValueList>
               <allowedValue>1</allowedValue>
               <allowedValue>2</allowedValue>
               <allowedValue>3</allowedValue>
            </allowedValueList>
         </stateVariable>
      </serviceStateTable>
   </scpd>
```