Linux SDK for UPnP Devices v1.2

Contents

1 Introduction 5
2 License 6
3 About Callbacks 7
4 The API 8
4.1 Error codes 8
4.1.1 UPNP_E_SUCCESS [0] 9
4.1.2 UPNP_E_INVALID_HANDLE [-100] 9
4.1.3 UPNP_E_INVALID_PARAM [-101] 10
4.1.4 UPNP_E_OUTOF_HANDLE [-102] 10
4.1.5 UPNP_E_OUTOF_MEMORY [-104] 10
4.1.6 UPNP_E_INIT [-105] 10
4.1.7 UPNP_E_INVALID_DESC [-107] 10
4.1.8 UPNP_E_INVALID_URL [-108] 11
4.1.9 UPNP_E_INVALID_SERVICE [-111] 11
4.1.10 UPNP_E_BAD_RESPONSE [-113] 11
4.1.11 UPNP_E_INVALID_ACTION [-115] 11
4.1.12 UPNP_E_FINISH [-116] 11
4.1.13 UPNP_E_INIT_FAILED [-117] 12
4.1.14 UPNP_E_BAD_HTTPMSG [-119] 12
4.1.15 UPNP_E_ALREADY_REGISTERED [-120] 12
4.1.16 UPNP_E_NETWORK_ERROR [-200] 12
4.1.17 UPNP_E_SOCKET_WRITE [-201] 12
4.1.18 UPNP_E_SOCKET_READ [-202] 13
4.1.19 UPNP_E_SOCKET_BIND [-203] 13
4.1.20 UPNP_E_SOCKET_CONNECT [-204] 13
4.1.21 UPNP_E_OUTOF_SOCKET [-205] 13
4.1.22 UPNP_E_LISTEN [-206] 13
4.1.23 UPNP_E_TIMEDOUT [-207] 14
4.1.24 UPNP_E_SOCKET_ERROR [-208] 14
4.1.25 UPNP_E_SUBSCRIBE_UNACCEPTED [-301] 14
4.1.26 UPNP_E_UNSUBSCRIBE_UNACCAPTED [-302] 14
4.1.27 UPNP_E_NOTIFY_UNACCEPTED [-303] 14
4.1.28 UPNP_E_INVALID_ARGUMENT [-501] 15
4.1.29 UPNP_E_FILE_NOT_FOUND [-502] 15
4.1.30 UPNP_E_FILE_READ_ERROR [-503] 15
4.1.31 UPNP_E_EXT_NOT_XML [-504] 15
4.1.32 UPNP_E_NOT_FOUND [-507] 15
4.1.33 UPNP_E_INTERNAL_ERROR [-911] 15
4.2 Constants, Structures, and Types 16
4.2.3 UPnP_EventType — *The reason code for an event callback.* 17
4.2.5 Upnp_SType — *Represents the different types of searches that can be performed using the SDK for UPnP Devices API.* 22
4.2.6 Upnp_DescType — *Specifies the type of description in* **UpnpRegisterRootDevice2**. 24
4.2.7 Upnp_Action_Request — *Returned as part of a* **UPNP_CONTROL_ACTION_COMPLETE** *callback.* 25
4.2.8 Upnp_State_Var_Request — *Represents the request for current value of a state variable in a service state table.* 28

4.2.9 Upnp_State_Var_Complete — *Represents the reply for the current value of a state variable in an asynchronous call.* 30
4.2.10 Upnp_Event — *Returned along with a* **UPNP_EVENT_RECEIVED** *callback.* 31
4.2.11 Upnp_Discovery — *Returned in a* **UPNP_DISCOVERY_RESULT** *callback.* 32
4.2.12 Upnp_Event_Subscribe — *Returned along with a* **UPNP_EVENT_SUBSCRIBE_COMPLETE** *or* **UPNP_EVENT_UNSUBSCRIBE_COMPLETE** *callback.* 35
4.2.13 Upnp_Subscription_Request — *Returned along with a* **UPNP_EVENT_SUBSCRIPTION_REQUEST** *callback.* 37
4.2.14 UpnpVirtualDirCallbacks — *The* **UpnpVirtualDirCallbacks** *structure contains the pointers to file-related callback functions a device application can register to virtualize URLs.* 38
4.3 Initialization and Registration 41
4.4 Discovery 50
4.5 Control 52
4.6 Eventing 59
4.7 Control Point HTTP API 77
4.8 Web Server API 86

5 Optional Tool APIs 91
6 Compile time configuration options 97
6.1 THREAD_IDLE_TIME 97
6.2 JOBS_PER_THREAD 97
6.3 MIN_THREADS 98
6.4 MAX_THREADS 98
6.5 HTTP_READ_BYTES 98
6.6 NUM_SSDP_COPY 98
6.7 SSDP_PAUSE 99
6.8 WEB_SERVER_BUF_SIZE 99
6.9 AUTO_RENEW_TIME 99
6.10 CP_MINIMUM_SUBSCRIPTION_TIME 99
6.11 MAX_SEARCH_TIME 99
6.12 MIN_SEARCH_TIME 100
6.13 AUTO_ADVERTISEMENT_TIME 100
6.14 SSDP_PACKET_DISTRIBUTE 100
6.15 Module Exclusion 100
6.16 DEBUG_LEVEL 101
6.17 DEBUG_TARGET 101

7 Other debugging features 102
7.1 DBGONLY 102

Linux SDK for UPnP Devices Version 1.2

Revision 1.2.1 (Sun Aug 14 20:27:14 2005)

1 Introduction

This document gives a brief description of the Linux SDK for UPnP Devices API. Section 1 covers the license under which the SDK is distributed. Section 2 talks about the callback functions used in many parts of the API. Finally, section 3 details the structures and functions that comprise the API.

The Linux SDK for UPnP Devices version 1.2 supports the following platforms:

- Linux* running on an Intel Architecture processor
- Linux running on an Intel StrongARM or XScale processor

* Other brands and names are the property of their respective owners.

2 License

3 About Callbacks

The Linux SDK for UPnP Devices contains functions that generate asynchronous callbacks. To simplify the application callback functions, these callbacks are executed on a thread owned by the SDK itself. The SDK executes the application's callback function in a thread context so the application can allocate memory and preserve the information it needs. The application can also use standard thread synchronization methods to ensure data integrity. Due to the possibility of deadlock, the application cannot call back into the SDK during these callbacks unless explicitly noted. There is no restriction in calling into the operating system or any other application interfaces.

4

The API

Names

4.1 **Error codes** 8
4.2 **Constants, Structures, and Types** 16
4.3 **Initialization and Registration** 41
4.4 **Discovery** 50
4.5 **Control** 52
4.6 **Eventing** 59
4.7 **Control Point HTTP API** 77
4.8 **Web Server API** 86

4.1

Error codes

Names

4.1.1 **UPNP_E_SUCCESS [0]** 9
4.1.2 **UPNP_E_INVALID_HANDLE [-100]** 9
4.1.3 **UPNP_E_INVALID_PARAM [-101]** 10
4.1.4 **UPNP_E_OUTOF_HANDLE [-102]** 10
4.1.5 **UPNP_E_OUTOF_MEMORY [-104]** 10
4.1.6 **UPNP_E_INIT [-105]** 10
4.1.7 **UPNP_E_INVALID_DESC [-107]** 10
4.1.8 **UPNP_E_INVALID_URL [-108]** 11
4.1.9 **UPNP_E_INVALID_SERVICE [-111]** 11
4.1.10 **UPNP_E_BAD_RESPONSE [-113]** 11
4.1.11 **UPNP_E_INVALID_ACTION [-115]** 11
4.1.12 **UPNP_E_FINISH [-116]** 11
4.1.13 **UPNP_E_INIT_FAILED [-117]** 12
4.1.14 **UPNP_E_BAD_HTTPMSG [-119]** 12
4.1.15 **UPNP_E_ALREADY_REGISTERED [-120]** 12
4.1.16 **UPNP_E_NETWORK_ERROR [-200]** 12
4.1.17 **UPNP_E_SOCKET_WRITE [-201]** 12
4.1.18 **UPNP_E_SOCKET_READ [-202]** 13

4.1.19 **UPNP_E_SOCKET_BIND [-203]** 13
4.1.20 **UPNP_E_SOCKET_CONNECT [-204]** 13
4.1.21 **UPNP_E_OUTOF_SOCKET [-205]** 13
4.1.22 **UPNP_E_LISTEN [-206]** 13
4.1.23 **UPNP_E_TIMEDOUT [-207]** 14
4.1.24 **UPNP_E_SOCKET_ERROR [-208]** 14
4.1.25 **UPNP_E_SUBSCRIBE_UNACCEPTED [-301]** 14
4.1.26 **UPNP_E_UNSUBSCRIBE_UNACCAPTED [-302]** 14
4.1.27 **UPNP_E_NOTIFY_UNACCEPTED [-303]** 14
4.1.28 **UPNP_E_INVALID_ARGUMENT [-501]** 15
4.1.29 **UPNP_E_FILE_NOT_FOUND [-502]** 15
4.1.30 **UPNP_E_FILE_READ_ERROR [-503]** 15
4.1.31 **UPNP_E_EXT_NOT_XML [-504]** 15
4.1.32 **UPNP_E_NOT_FOUND [-507]** 15
4.1.33 **UPNP_E_INTERNAL_ERROR [-911]** 15

The functions in the SDK API can return a variety of error codes to describe problems encountered during execution. This section lists the error codes and provides a brief description of what each error code means. Refer to the documentation for each function for a description of what an error code means in that context.

4.1.1

UPNP_E_SUCCESS [0]

`UPNP_E_SUCCESS` signifies that the operation completed successfully. For asynchronous functions, this only means that the packet generated by the operation was successfully transmitted on the network. The result of the entire operation comes as part of the callback for that operation.

4.1.2

UPNP_E_INVALID_HANDLE [-100]

`UPNP_E_INVALID_HANDLE` signifies that the handle passed to a function is not a recognized as a valid handle.

4.1.3

UPNP_E_INVALID_PARAM [-101]

UPNP_E_INVALID_PARAM signifies that one or more of the parameters passed to the function is not valid. Refer to the documentation for each function for more information on the valid ranges of the parameters.

4.1.4

UPNP_E_OUTOF_HANDLE [-102]

UPNP_E_OUTOF_HANDLE signifies that the SDK does not have any more space for additional handles. The SDK allocates space for only a few handles in order to conserve memory.

4.1.5

UPNP_E_OUTOF_MEMORY [-104]

UPNP_E_OUTOF_MEMORY signifies that not enough resources are currently available to complete the operation. Most operations require some free memory in order to complete their work.

4.1.6

UPNP_E_INIT [-105]

UPNP_E_INIT signifies that the SDK has already been initialized. The SDK needs to be initialied only once per process. Any additional initialization attempts simply return this error with no other ill effects.

4.1.7

UPNP_E_INVALID_DESC [-107]

UPNP_E_INVALID_DESC signifies that the description document passed to **UpnpRegisterRootDevice** or **UpnpRegisterRootDevice2** is an invalid description document.

4.1.8

UPNP_E_INVALID_URL [-108]

`UPNP_E_INVALID_URL` signifies that a URL passed into the function is invalid. The actual cause is function specific, but in general, the URL itself might be malformed (e.g. have invalid characters in it) or the host might be unreachable.

4.1.9

UPNP_E_INVALID_SERVICE [-111]

`UPNP_E_INVALID_SERVICE` is returned only by **UpnpNotify**, **UpnpNotifyExt**, **UpnpAcceptSubscription**, and **UpnpAcceptSubscriptionExt** to signify that the device ID/service ID pair does not refer to a valid service.

4.1.10

UPNP_E_BAD_RESPONSE [-113]

`UPNP_E_BAD_RESPONSE` signifies that the response received from the remote side of a connection is not correct for the protocol. This applies to the GENA, SOAP, and HTTP protocols.

4.1.11

UPNP_E_INVALID_ACTION [-115]

`UPNP_E_INVALID_ACTION` signifies that the SOAP action message is invalid. This can be because the DOM document passed to the function was malformed or the action message is not correct for the given action.

4.1.12

UPNP_E_FINISH [-116]

`UPNP_E_FINISH` signifies that **UpnpInit** has not been called, or that **UpnpFinish** has already been called. None of the API functions operate until **UpnpInit** successfully completes.

4.1.13

UPNP_E_INIT_FAILED [-117]

UPNP_E_INIT_FAILED signifies that **UpnpInit** cannot complete. The typical reason is failure to allocate sufficient resources.

4.1.14

UPNP_E_BAD_HTTPMSG [-119]

UPNP_E_BAD_HTTPMSG signifies that the HTTP message contains invalid message headers. The error always refers to the HTTP message header received from the remote host. The main areas where this occurs are in SOAP control messages (e.g. **UpnpSendAction**), GENA subscription message (e.g. **UpnpSubscribe**), GENA event notifications (e.g. **UpnpNotify**), and HTTP transfers (e.g. **UpnpDownloadXmlDoc**).

4.1.15

UPNP_E_ALREADY_REGISTERED [-120]

UPNP_E_ALREADY_REGISTERED signifies that a client or a device is already registered. The SDK currently has a limit of one registered client and one registered device per process.

4.1.16

UPNP_E_NETWORK_ERROR [-200]

UPNP_E_NETWORK_ERROR signifies that a network error occurred. It is the generic error code for network problems that are not covered under one of the more specific error codes. The typical meaning is the SDK failed to read the local IP address or had problems configuring one of the sockets.

4.1.17

UPNP_E_SOCKET_WRITE [-201]

UPNP_E_SOCKET_WRITE signifies an error writing to a socket. This occurs in any function that makes network connections, such as discovery (e.g. **UpnpSearchAsync** or **UpnpSendAdvertisement**), control (e.g. **UpnpSendAction**), eventing (e.g. **UpnpNotify**), and HTTP functions (e.g. **UpnpDownloadXmlDoc**).

4.1.18

UPNP_E_SOCKET_READ [-202]

UPNP_E_SOCKET_READ signifies an error reading from a socket. This occurs in any function that makes network connections, such as discovery (e.g. **UpnpSearchAsync** or **UpnpSendAdvertisement**), control (e.g. **UpnpSendAction**), eventing (e.g. **UpnpNotify**), and HTTP functions (e.g. **UpnpDownloadXmlDoc**).

4.1.19

UPNP_E_SOCKET_BIND [-203]

UPNP_E_SOCKET_BIND signifies that the SDK had a problem binding a socket to a network interface. This occurs in any function that makes network connections, such as discovery (e.g. **UpnpSearchAsync** or **UpnpSendAdvertisement**), control (e.g. **UpnpSendAction**), eventing (e.g. **UpnpNotify**), and HTTP functions (e.g. **UpnpDownloadXmlDoc**).

4.1.20

UPNP_E_SOCKET_CONNECT [-204]

UPNP_E_SOCKET_CONNECT signifies that the SDK had a problem connecting to a remote host. This occurs in any function that makes network connections, such as discovery (e.g. **UpnpSearchAsync** or **UpnpSendAdvertisement**), control (e.g. **UpnpSendAction**), eventing (e.g. **UpnpNotify**), and HTTP functions (e.g. **UpnpDownloadXmlDoc**).

4.1.21

UPNP_E_OUTOF_SOCKET [-205]

UPNP_E_OUTOF_SOCKET signifies that the SDK cannot create any more sockets. This occurs in any function that makes network connections, such as discovery (e.g. **UpnpSearchAsync** or **UpnpSendAdvertisement**), control (e.g. **UpnpSendAction**), eventing (e.g. **UpnpNotify**), and HTTP functions (e.g. **UpnpDownloadXmlDoc**).

4.1.22

UPNP_E_LISTEN [-206]

UPNP_E_LISTEN signifies that the SDK had a problem setting the socket to listen for incoming connections. This error only happens during initialization (i.e. **UpnpInit**).

4.1.23

UPNP_E_TIMEDOUT [-207]

UPNP_E_TIMEDOUT signifies that too much time elapsed before the required number of bytes were sent or received over a socket. This error can be returned by any function that performs network operations.

4.1.24

UPNP_E_SOCKET_ERROR [-208]

UPNP_E_SOCKET_ERROR is the generic socket error code for conditions not covered by other error codes. This error can be returned by any functions that performs network operations.

4.1.25

UPNP_E_SUBSCRIBE_UNACCEPTED [-301]

UPNP_E_SUBSCRIBE_UNACCEPTED signifies that a subscription request was rejected from the remote side.

4.1.26

UPNP_E_UNSUBSCRIBE_UNACCAPTED [-302]

UPNP_E_UNSUBSCRIBE_UNACCEPTED signifies that an unsubscribe request was rejected from the remote side.

4.1.27

UPNP_E_NOTIFY_UNACCEPTED [-303]

UPNP_E_NOTIFY_UNACCEPTED signifies that the remote host did not accept the notify sent from the local device.

4.1.28

UPNP_E_INVALID_ARGUMENT [-501]

UPNP_E_INVALID_ARGUMENT signifies that one or more of the parameters passed to a function is invalid. Refer to the individual function descriptions for the acceptable ranges for parameters.

4.1.29

UPNP_E_FILE_NOT_FOUND [-502]

UPNP_E_FILE_NOT_FOUND signifies that the filename passed to one of the device registration functions was not found or was not accessible.

4.1.30

UPNP_E_FILE_READ_ERROR [-503]

UPNP_E_FILE_READ_ERROR signifies an error when reading a file.

4.1.31

UPNP_E_EXT_NOT_XML [-504]

UPNP_E_EXT_NOT_XML signifies that the file name of the description document passed to **UpnpRegisterRootDevice2** does not end in ".xml".

4.1.32

UPNP_E_NOT_FOUND [-507]

UPNP_E_NOT_FOUND signifies that the response to a SOAP request did not contain the required XML constructs.

4.1.33

UPNP_E_INTERNAL_ERROR [-911]

UPNP_E_INTERNAL_ERROR is the generic error code for internal conditions not covered by other error codes.

4.2 Constants, Structures, and Types

Names

4.2.1	typedef int	**UpnpClient_Handle**	*Returned when a control point application registers with* **UpnpRegisterClient**. ..	17
4.2.2	typedef int	**UpnpDevice_Handle**	*Returned when a device application registers with* **UpnpRegisterRootDevice** *or* **UpnpRegisterRootDevice2**.	17
4.2.3	enum	**UPnP_EventType**	*The reason code for an event callback.*	17
4.2.4	typedef char	**Upnp_SID[44]**	*The* **Upnp_SID** *holds the subscription identifier for a subscription between a client and a device.*	22
4.2.5	enum	**Upnp_SType**	*Represents the different types of searches that can be performed using the SDK for UPnP Devices API.*	22
4.2.6	enum	**Upnp_DescType**	*Specifies the type of description in* **UpnpRegisterRootDevice2**.	24
4.2.7	struct	**Upnp_Action_Request**	*Returned as part of a* **UPNP_CONTROL_ACTION_COMPLETE** *callback.*	25
4.2.8	struct	**Upnp_State_Var_Request**	*Represents the request for current value of a state variable in a service state table.*	28
4.2.9	struct	**Upnp_State_Var_Complete**	*Represents the reply for the current value of a state variable in an asynchronous call.*	30
4.2.10	struct	**Upnp_Event**	*Returned along with a* **UPNP_EVENT_RECEIVED** *callback.*	31
4.2.11	struct	**Upnp_Discovery**	*Returned in a* **UPNP_DISCOVERY_RESULT** *callback.*	32
4.2.12	struct	**Upnp_Event_Subscribe**	*Returned along with a* **UPNP_EVENT_SUBSCRIBE_COMPLETE** *or* **UPNP_EVENT_UNSUBSCRIBE_COMPLETE** *callback.*	35
4.2.13	struct	**Upnp_Subscription_Request**	*Returned along with a* **UPNP_EVENT_SUBSCRIPTION_REQUEST** *callback.*	37
4.2.14	struct	**UpnpVirtualDirCallbacks**		

The **UpnpVirtualDirCallbacks** *structure contains the pointers to file-related callback functions a device application can register to virtualize URLs.* 38

4.2.15 typedef int **(*Upnp_FunPtr)** (IN Upnp_EventType EventType, IN void* Event, IN void* Cookie)
All callback functions share the same prototype, documented below. 41

4.2.1

typedef int **UpnpClient_Handle**

Returned when a control point application registers with **UpnpRegisterClient**.

Returned when a control point application registers with **UpnpRegisterClient**. Client handles can only be used with functions that operate with a client handle.

4.2.2

typedef int **UpnpDevice_Handle**

Returned when a device application registers with **UpnpRegisterRootDevice** *or* **UpnpRegisterRootDevice2**.

Returned when a device application registers with **UpnpRegisterRootDevice** or **UpnpRegisterRootDevice2**. Device handles can only be used with functions that operate with a device handle.

4.2.3

enum **UPnP_EventType**

The reason code for an event callback.

Names

4.2.3.1 **UPNP_CONTROL_ACTION_REQUEST**
Received by a device when a control point issues a control request. 19

4.2.3.2 **UPNP_CONTROL_ACTION_COMPLETE**

A **UpnpSendActionAsync** *call completed.* 19

4.2.3.3 **UPNP_CONTROL_GET_VAR_REQUEST**
Received by a device when a query for a single service variable arrives. 19

4.2.3.4 **UPNP_CONTROL_GET_VAR_COMPLETE**
A **UpnpGetServiceVarStatus** *call completed.* 19

4.2.3.5 **UPNP_DISCOVERY_ADVERTISEMENT_ALIVE**
Received by a control point when a new device or service is available. 19

4.2.3.6 **UPNP_DISCOVERY_ADVERTISEMENT_BYEBYE**
Received by a control point when a device or service shuts down. 20

4.2.3.7 **UPNP_DISCOVERY_SEARCH_RESULT**
Received by a control point when a matching device or service responds. 20

4.2.3.8 **UPNP_DISCOVERY_SEARCH_TIMEOUT**
Received by a control point when the search timeout expires. 20

4.2.3.9 **UPNP_EVENT_SUBSCRIPTION_REQUEST**
Received by a device when a subscription arrives. 20

4.2.3.10 **UPNP_EVENT_RECEIVED**
Received by a control point when an event arrives. 21

4.2.3.11 **UPNP_EVENT_RENEWAL_COMPLETE**
A **UpnpRenewSubscriptionAsync** *call completed.* 21

4.2.3.12 **UPNP_EVENT_SUBSCRIBE_COMPLETE**
A **UpnpSubscribeAsync** *call completed.* 21

4.2.3.13 **UPNP_EVENT_UNSUBSCRIBE_COMPLETE**
A **UpnpUnSubscribeAsync** *call completed.* 21

4.2.3.14 **UPNP_EVENT_AUTORENEWAL_FAILED**
The auto-renewal of a client subscription failed. 22

4.2.3.15 **UPNP_EVENT_SUBSCRIPTION_EXPIRED**
A client subscription has expired. 22

The **Event** parameter will be different depending on the reason for the callback. The descriptions for each event type describe the contents of the **Event** parameter.

4.2.3.1

UPNP_CONTROL_ACTION_REQUEST

Received by a device when a control point issues a control request.

Received by a device when a control point issues a control request. The **Event** parameter contains a pointer to a **Upnp_Action_Request** structure containing the action. The application stores the results of the action in this structure.

4.2.3.2 UPNP_CONTROL_ACTION_COMPLETE

A **UpnpSendActionAsync** *call completed.*

A **UpnpSendActionAsync** call completed. The **Event** parameter contains a pointer to a **Upnp_Action_Complete** structure with the results of the action.

4.2.3.3 UPNP_CONTROL_GET_VAR_REQUEST

Received by a device when a query for a single service variable arrives.

Received by a device when a query for a single service variable arrives. The **Event** parameter contains a pointer to a **Upnp_State_Var_Request** structure containing the name of the variable and value.

4.2.3.4 UPNP_CONTROL_GET_VAR_COMPLETE

A **UpnpGetServiceVarStatus** *call completed.*

A **UpnpGetServiceVarStatus** call completed. The **Event** parameter contains a pointer to a **Upnp_State_Var_Complete** structure containing the value for the variable.

4.2.3.5 UPNP_DISCOVERY_ADVERTISEMENT_ALIVE

Received by a control point when a new device or service is available.

Received by a control point when a new device or service is available. The **Event** parameter contains a pointer to a **Upnp_Discovery** structure with the information about the device or service.

4.2.3.6 UPNP_DISCOVERY_ADVERTISEMENT_BYEBYE

Received by a control point when a device or service shuts down.

Received by a control point when a device or service shuts down. The **Event** parameter contains a pointer to a **Upnp_Discovery** structure containing the information about the device or service.

4.2.3.7 UPNP_DISCOVERY_SEARCH_RESULT

Received by a control point when a matching device or service responds.

Received by a control point when a matching device or service responds. The **Event** parameter contains a pointer to a **Upnp_Discovery** structure containing the information about the reply to the search request.

4.2.3.8 UPNP_DISCOVERY_SEARCH_TIMEOUT

Received by a control point when the search timeout expires.

Received by a control point when the search timeout expires. The SDK generates no more callbacks for this search after this event. The **Event** parameter is `NULL`.

4.2.3.9 UPNP_EVENT_SUBSCRIPTION_REQUEST

Received by a device when a subscription arrives.

Received by a device when a subscription arrives. The **Event** parameter contains a pointer to a **Upnp_Subscription_Request** structure. At this point, the subscription has already been accepted. **UpnpAcceptSubscription** needs to be called to confirm the subscription and transmit the initial state table. This can be done during this callback. The SDK generates no events for a subscription unless the device application calls **UpnpAcceptSubscription**.

4.2.3.10

UPNP_EVENT_RECEIVED

Received by a control point when an event arrives.

Received by a control point when an event arrives. The **Event** parameter contains a **Upnp_Event** structure with the information about the event.

4.2.3.11

UPNP_EVENT_RENEWAL_COMPLETE

A **UpnpRenewSubscriptionAsync** *call completed.*

A **UpnpRenewSubscriptionAsync** call completed. The status of the renewal is in the **Event** parameter as a **Upnp_Event_Subscription** structure.

4.2.3.12

UPNP_EVENT_SUBSCRIBE_COMPLETE

A **UpnpSubscribeAsync** *call completed.*

A **UpnpSubscribeAsync** call completed. The status of the subscription is in the **Event** parameter as a **Upnp_Event_Subscription** structure.

4.2.3.13

UPNP_EVENT_UNSUBSCRIBE_COMPLETE

A **UpnpUnSubscribeAsync** *call completed.*

A **UpnpUnSubscribeAsync** call completed. The status of the subscription is in the **Event** parameter as a **Upnp_Event_Subscribe** structure.

4.2.3.14

UPNP_EVENT_AUTORENEWAL_FAILED

The auto-renewal of a client subscription failed.

The auto-renewal of a client subscription failed. The **Event** parameter is a **Upnp_Event_Subscribe** structure with the error code set appropriately. The subscription is no longer valid.

4.2.3.15

UPNP_EVENT_SUBSCRIPTION_EXPIRED

A client subscription has expired.

A client subscription has expired. This will only occur if auto-renewal of subscriptions is disabled. The **Event** parameter is a **Upnp_Event_Subscribe** structure. The subscription is no longer valid.

4.2.4

typedef char **Upnp_SID**[**44**]

The **Upnp_SID** *holds the subscription identifier for a subscription between a client and a device.*

The **Upnp_SID** holds the subscription identifier for a subscription between a client and a device. The SID is a string representation of a globally unique id (GUID) and should not be modified.

4.2.5

enum **Upnp_SType**

Represents the different types of searches that can be performed using the SDK for UPnP Devices API.

Names

4.2.5.1	**UPNP_S_ALL**	*Search for all devices and services on the network.*	23
4.2.5.2	**UPNP_S_ROOT**	*Search for all root devices on the network.*	23
4.2.5.3	**UPNP_S_DEVICE**	*Search for a particular device type or a particular device instance.*	23
4.2.5.4	**UPNP_S_SERVICE**	*Search for a particular service type, possibly on a particular device type or device instance.*	24

By specifying these different values to **UpnpSearchAsync**, the control point application can control the scope of the search from all devices to specific devices or services.

4.2.5.1

UPNP_S_ALL

Search for all devices and services on the network.

Search for all devices and services on the network.

4.2.5.2

UPNP_S_ROOT

Search for all root devices on the network.

Search for all root devices on the network.

4.2.5.3

UPNP_S_DEVICE

Search for a particular device type or a particular device instance.

Search for a particular device type or a particular device instance.

4.2.5.4

UPNP_S_SERVICE

Search for a particular service type, possibly on a particular device type or device instance.

Search for a particular service type, possibly on a particular device type or device instance.

4.2.6

enum Upnp_DescType

Specifies the type of description in **UpnpRegisterRootDevice2**.

Names

4.2.6.1 **UPNPREG_URL_DESC**
The description is the URL to the description document. 24

4.2.6.2 **UPNPREG_FILENAME_DESC**
The description is a file name on the local file system containing the description of the device. 25

4.2.6.3 **UPNPREG_BUF_DESC**
The description is a pointer to a character array containing the XML description document. 25

These values control how **UpnpRegisterRootDevice2** interprets the **description** parameter.

4.2.6.1

UPNPREG_URL_DESC

The description is the URL to the description document.

The description is the URL to the description document.

4.2.6.2

UPNPREG_FILENAME_DESC

The description is a file name on the local file system containing the description of the device.

The description is a file name on the local file system containing the description of the device.

4.2.6.3

UPNPREG_BUF_DESC

The description is a pointer to a character array containing the XML description document.

The description is a pointer to a character array containing the XML description document.

4.2.7

struct Upnp_Action_Request

Returned as part of a **UPNP_CONTROL_ACTION_COMPLETE** *callback.*

Members

4.2.7.1 int **ErrCode** *The result of the operation.* 26

4.2.7.2 int **Socket** *The socket number of the connection to the requestor.* 26

4.2.7.3 char **ErrStr** [LINE_SIZE] *The error string in case of error.* 26

4.2.7.4 char **ActionName** [NAME_SIZE] *The Action Name.* 26

4.2.7.5 char **DevUDN** [NAME_SIZE] *The unique device ID.* 27

4.2.7.6 char **ServiceID** [NAME_SIZE] *The service ID.* 27

4.2.7.7 IXML_Document* **ActionRequest** *The DOM document describing the action.* 27

4.2.7.8 IXML_Document* **ActionResult** *The DOM document describing the result of the action.* 27

4.2.7.9 struct in_addr

CtrlPtIPAddr *IP address of the control point requesting this action.* 27

4.2.7.10 IXML_Document*

SoapHeader *The DOM document containing the information from the the SOAP header.* 28

Returned as part of a **UPNP_CONTROL_ACTION_COMPLETE** callback.

4.2.7.1

int **ErrCode**

The result of the operation.

The result of the operation.

4.2.7.2

int **Socket**

The socket number of the connection to the requestor.

The socket number of the connection to the requestor.

4.2.7.3

char **ErrStr** [LINE_SIZE]

The error string in case of error.

The error string in case of error.

4.2.7.4

char **ActionName** [NAME_SIZE]

The Action Name.

The Action Name.

4.2.7.5

char **DevUDN** [NAME_SIZE]

The unique device ID.

The unique device ID.

4.2.7.6

char **ServiceID** [NAME_SIZE]

The service ID.

The service ID.

4.2.7.7

IXML_Document* **ActionRequest**

The DOM document describing the action.

The DOM document describing the action.

4.2.7.8

IXML_Document* **ActionResult**

The DOM document describing the result of the action.

The DOM document describing the result of the action.

4.2.7.9

struct in_addr **CtrlPtIPAddr**

IP address of the control point requesting this action.

IP address of the control point requesting this action.

4.2.7.10

IXML_Document* **SoapHeader**

The DOM document containing the information from the the SOAP header.

The DOM document containing the information from the the SOAP header.

4.2.8

struct **Upnp_State_Var_Request**

Represents the request for current value of a state variable in a service state table.

Members

4.2.8.1	int	**ErrCode**	*The result of the operation.*	28
4.2.8.2	int	**Socket**	*The socket number of the connection to the requestor.*	29
4.2.8.3	char	**ErrStr** [LINE_SIZE]	*The error string in case of error.*	29
4.2.8.4	char	**DevUDN** [NAME_SIZE]	*The unique device ID.*	29
4.2.8.5	char	**ServiceID** [NAME_SIZE]	*The service ID.*	29
4.2.8.6	char	**StateVarName** [NAME_SIZE]	*The name of the variable.*	29
4.2.8.7	struct in_addr	**CtrlPtIPAddr**	*IP address of sender requesting the state variable.*	30
4.2.8.8	DOMString	**CurrentVal**	*The current value of the variable.*	30

Represents the request for current value of a state variable in a service state table.

4.2.8.1

int **ErrCode**

The result of the operation.

The result of the operation.

4.2.8.2

int **Socket**

The socket number of the connection to the requestor.

The socket number of the connection to the requestor.

4.2.8.3

char **ErrStr** [LINE_SIZE]

The error string in case of error.

The error string in case of error.

4.2.8.4

char **DevUDN** [NAME_SIZE]

The unique device ID.

The unique device ID.

4.2.8.5

char **ServiceID** [NAME_SIZE]

The service ID.

The service ID.

4.2.8.6

char **StateVarName** [NAME_SIZE]

The name of the variable.

The name of the variable.

4.2.8.7

struct in_addr **CtrlPtIPAddr**

IP address of sender requesting the state variable.

IP address of sender requesting the state variable.

4.2.8.8

DOMString **CurrentVal**

The current value of the variable.

The current value of the variable. This needs to be allocated by the caller. When finished with it, the SDK frees this **DOMString**.

4.2.9

struct **Upnp_State_Var_Complete**

Represents the reply for the current value of a state variable in an asynchronous call.

Members

4.2.9.1	int	**ErrCode**	*The result of the operation.*	30
4.2.9.2	char	**CtrlUrl** [NAME_SIZE]	*The control URL for the service.*	31
4.2.9.3	char	**StateVarName** [NAME_SIZE]	*The name of the variable.*	31
4.2.9.4	DOMString	**CurrentVal**	*The current value of the variable or error string in case of error.*	31

Represents the reply for the current value of a state variable in an asynchronous call.

4.2.9.1

int **ErrCode**

The result of the operation.

The result of the operation.

4.2.9.2

char **CtrlUrl** [NAME_SIZE]

The control URL for the service.

The control URL for the service.

4.2.9.3

char **StateVarName** [NAME_SIZE]

The name of the variable.

The name of the variable.

4.2.9.4

DOMString **CurrentVal**

The current value of the variable or error string in case of error.

The current value of the variable or error string in case of error.

4.2.10

struct **Upnp_Event**

Returned along with a **UPNP_EVENT_RECEIVED** *callback.*

Members

4.2.10.1 Upnp_SID **Sid** *The subscription ID for this subscription.* 32

4.2.10.2 int **EventKey** *The event sequence number.* 32

4.2.10.3 IXML_Document* **ChangedVariables** *The DOM tree representing the changes generating the event.* 32

Returned along with a **UPNP_EVENT_RECEIVED** callback.

4.2.10.1

Upnp_SID **Sid**

The subscription ID for this subscription.

The subscription ID for this subscription.

4.2.10.2

int **EventKey**

The event sequence number.

The event sequence number.

4.2.10.3

IXML_Document* **ChangedVariables**

The DOM tree representing the changes generating the event.

The DOM tree representing the changes generating the event.

4.2.11

struct **Upnp_Discovery**

Returned in a **UPNP_DISCOVERY_RESULT** *callback.*

Members

4.2.11.1 int **ErrCode** *The result code of the* **UpnpSearchAsync** *call.* 33

4.2.11.2 int **Expires** *The expiration time of the advertisement.* 33

4.2.11.3 char **DeviceId** [LINE_SIZE] *The unique device identifier.* 33

4.2.11.4 char **DeviceType** [LINE_SIZE] *The device type.* 34

4.2.11.5 char **ServiceType** [LINE_SIZE]

The service type. 34

4.2.11.6 char **ServiceVer** [LINE_SIZE] *The service version.* 34

4.2.11.7 char **Location** [LINE_SIZE] *The URL to the UPnP description document for the device.* 34

4.2.11.8 char **Os** [LINE_SIZE] *The operating system the device is running.* 34

4.2.11.9 char **Date** [LINE_SIZE] *Date when the response was generated.* . 35

4.2.11.10 char **Ext** [LINE_SIZE] *Confirmation that the MAN header was understood by the device.* 35

4.2.11.11 SOCKADDRIN* **DestAddr** *The host address of the device responding to the search.* 35

Returned in a **UPNP_DISCOVERY_RESULT** callback.

4.2.11.1

int **ErrCode**

The result code of the **UpnpSearchAsync** *call.*

The result code of the **UpnpSearchAsync** call.

4.2.11.2

int **Expires**

The expiration time of the advertisement.

The expiration time of the advertisement.

4.2.11.3

char **DeviceId** [LINE_SIZE]

The unique device identifier.

The unique device identifier.

4.2.11.4

char **DeviceType** [LINE_SIZE]

The device type.

The device type.

4.2.11.5

char **ServiceType** [LINE_SIZE]

The service type.

The service type.

4.2.11.6

char **ServiceVer** [LINE_SIZE]

The service version.

The service version.

4.2.11.7

char **Location** [LINE_SIZE]

The URL to the UPnP description document for the device.

The URL to the UPnP description document for the device.

4.2.11.8

char **Os** [LINE_SIZE]

The operating system the device is running.

The operating system the device is running.

4.2.11.9

char **Date** [LINE_SIZE]

Date when the response was generated.

Date when the response was generated.

4.2.11.10

char **Ext** [LINE_SIZE]

Confirmation that the MAN header was understood by the device.

Confirmation that the MAN header was understood by the device.

4.2.11.11

SOCKADDRIN* **DestAddr**

The host address of the device responding to the search.

The host address of the device responding to the search.

4.2.12

struct **Upnp_Event_Subscribe**

Returned along with a **UPNP_EVENT_SUBSCRIBE_COMPLETE** *or* **UPNP_EVENT_UNSUBSCRIBE_COMPLETE** *callback.*

Members

4.2.12.1	Upnp_SID	**Sid**	*The SID for this subscription.*	36
4.2.12.2	int	**ErrCode**	*The result of the operation.*	36
4.2.12.3	char	**PublisherUrl** [NAME_SIZE]	*The event URL being subscribed to or removed from.*	36
4.2.12.4	int	**TimeOut**	*The actual subscription time (for subscriptions only).*	36

Returned along with a **UPNP_EVENT_SUBSCRIBE_COMPLETE** or **UPNP_EVENT_UNSUBSCRIBE_COMPLETE** callback.

4.2.12.1

Upnp_SID **Sid**

The SID for this subscription.

The SID for this subscription. For subscriptions, this only contains a valid SID if the **Upnp_EventSubscribe.result** field contains a `UPNP_E_SUCCESS` result code. For unsubscriptions, this contains the SID from which the subscription is being unsubscribed.

4.2.12.2

int **ErrCode**

The result of the operation.

The result of the operation.

4.2.12.3

char **PublisherUrl** [NAME_SIZE]

The event URL being subscribed to or removed from.

The event URL being subscribed to or removed from.

4.2.12.4

int **TimeOut**

The actual subscription time (for subscriptions only).

The actual subscription time (for subscriptions only).

4.2.13

struct **Upnp_Subscription_Request**

Returned along with a **UPNP_EVENT_SUBSCRIPTION_REQUEST** *callback.*

Members

4.2.13.1	char*	**ServiceId**	*The identifier for the service being subscribed to.*	37
4.2.13.2	char*	**UDN**	*Universal device name.*	37
4.2.13.3	Upnp_SID	**Sid**	*The assigned subscription ID for this subscription.*	37

Returned along with a **UPNP_EVENT_SUBSCRIPTION_REQUEST** callback.

4.2.13.1

char* **ServiceId**

The identifier for the service being subscribed to.

The identifier for the service being subscribed to.

4.2.13.2

char* **UDN**

Universal device name.

Universal device name.

4.2.13.3

Upnp_SID **Sid**

The assigned subscription ID for this subscription.

The assigned subscription ID for this subscription.

4.2.14

struct **UpnpVirtualDirCallbacks**

The **UpnpVirtualDirCallbacks** *structure contains the pointers to file-related callback functions a device application can register to virtualize URLs.*

Members

4.2.14.1 int **(*get_info)** (IN const char* filename, OUT struct File_Info* info) *Called by the web server to query information on a file.* 38

4.2.14.2 UpnpWebFileHandle **(*open)** (IN const char* filename, IN enum UpnpOpenFileMode Mode) *Called by the web server to open a file.* . 39

4.2.14.3 int **(*read)** (IN UpnpWebFileHandle fileHnd, OUT char* buf, IN size_t buflen) *Called by the web server to perform a sequential read from an open file.* 39

4.2.14.4 int **(*write)** (IN UpnpWebFileHandle fileHnd, IN char* buf, IN size_t buflen) *Called by the web server to perform a sequential write to an open file.* 40

4.2.14.5 int **(*seek)** (IN UpnpWebFileHandle fileHnd, IN long offset, IN int origin) *Called by the web server to move the file pointer, or offset, into an open file.* ... 40

4.2.14.6 int **(*close)** (IN UpnpWebFileHandle fileHnd) *Called by the web server to close a file opened via the* **open** *callback.* 40

The **UpnpVirtualDirCallbacks** structure contains the pointers to file-related callback functions a device application can register to virtualize URLs.

4.2.14.1

int **(*get_info)** (IN const char* filename, OUT struct File_Info* info)

Called by the web server to query information on a file.

Called by the web server to query information on a file. The callback should return 0 on success or -1 on an error.

Parameters:

`filename`	The name of the file to query.
`info`	Pointer to a structure to store the information on the file.

4.2.14.2

UpnpWebFileHandle **(*open)** (IN const char* filename, IN enum UpnpOpenFileMode Mode)

Called by the web server to open a file.

Called by the web server to open a file. The callback should return a valid handle if the file can be opened. Otherwise, it should return `NULL` to signify an error.

Parameters:

`filename`	The name of the file to open.
`Mode`	The mode in which to open the file. Valid values are `UPNP_READ` or `UPNP_WRITE`.

4.2.14.3

int **(*read)** (IN UpnpWebFileHandle fileHnd, OUT char* buf, IN size_t buflen)

Called by the web server to perform a sequential read from an open file.

Called by the web server to perform a sequential read from an open file. The callback should copy **buflen** bytes from the file into the buffer.

Return Value: `[int]` An integer representing one of the following:

- `0`: The file contains no more data (EOF).
- `>0`: A successful read of the number of bytes in the return code.
- `<0`: An error occurred reading the file.

Parameters:

`fileHnd`	The handle of the file to read.
`buf`	The buffer in which to place the data.
`buflen`	The size of the buffer (i.e. the number of bytes to read).

4.2.14.4

int **(*write)** (IN UpnpWebFileHandle fileHnd, IN char* buf, IN size_t buflen)

Called by the web server to perform a sequential write to an open file.

Called by the web server to perform a sequential write to an open file. The callback should write **buflen** bytes into the file from the buffer. It should return the actual number of bytes written, which might be less than **buflen** in the case of a write error.

Parameters:

`fileHnd`	The handle of the file to write.
`buf`	The buffer with the bytes to write.
`buflen`	The number of bytes to write.

4.2.14.5

int **(*seek)** (IN UpnpWebFileHandle fileHnd, IN long offset, IN int origin)

Called by the web server to move the file pointer, or offset, into an open file.

Called by the web server to move the file pointer, or offset, into an open file. The **origin** parameter determines where to start moving the file pointer. A value of `SEEK_CUR` moves the file pointer relative to where it is. The **offset** parameter can be either positive (move forward) or negative (move backward). `SEEK_END` moves relative to the end of the file. A positive **offset** extends the file. A negative **offset** moves backward in the file. Finally, `SEEK_SET` moves to an absolute position in the file. In this case, **offset** must be positive. The callback should return 0 on a successful seek or a non-zero value on an error.

Parameters:

`fileHnd`	The handle of the file to move the file pointer.
`offset`	The number of bytes to move in the file. Positive values move foward and negative values move backward. Note that this must be positive if the **origin** is `SEEK_SET`.
`origin`	The position to move relative to. It can be `SEEK_CUR` to move relative to the current position, `SEEK_END` to move relative to the end of the file, or `SEEK_SET` to specify an absolute offset.

4.2.14.6

int **(*close)** (IN UpnpWebFileHandle fileHnd)

Called by the web server to close a file opened via the **open** *callback.*

Called by the web server to close a file opened via the **open** callback. It should return 0 on success, or a non-zero value on an error.

Parameters:

`fileHnd`	The handle of the file to close.

4.2.15

typedef int **(*Upnp_FunPtr)** (IN Upnp_EventType EventType, IN void* Event, IN void* Cookie)

All callback functions share the same prototype, documented below.

All callback functions share the same prototype, documented below. Note that any memory passed to the callback function is valid only during the callback and should be copied if it needs to persist. This callback function needs to be thread safe. The context of the callback is always on a valid thread context and standard synchronization methods can be used. Note, however, because of this the callback cannot call SDK functions unless explicitly noted.

```
int CallbackFxn( Upnp_EventType EventType, void* Event, void* Cookie );
```

where **EventType** is the event that triggered the callback, **Event** is a structure that denotes event-specific information for that event, and **Cookie** is the user data passed when the callback was registered.

See **Upnp_EventType** for more information on the callback values and the associated **Event** parameter.

The return value of the callback is currently ignored. It may be used in the future to communicate results back to the SDK.

4.3

Initialization and Registration

Names

4.3.1 int **UpnpInit** (IN const char* HostIP, IN unsigned short DestPort)
Initializes the Linux SDK for UPnP Devices. 42

4.3.2 int **UpnpFinish** ()
Terminates the Linux SDK for UPnP Devices. 43

4.3.3 unsigned short **UpnpGetServerPort** (void)
If '0' is used as the port number in **UpnpInit**, *then this function can be used to retrieve the actual port allocated to the SDK.* 44

4.3.4 char* **UpnpGetServerIpAddress** (void)

If `NULL` *is used as the IP address in* **UpnpInit**, *then this function can be used to retrieve the actual interface address on which device is running.* 44

4.3.5 int **UpnpRegisterClient** (IN Upnp_FunPtr Callback,
IN const void* Cookie,
OUT UpnpClient_Handle* Hnd)
UpnpRegisterClient *registers a control point application with the SDK.* 45

4.3.6 int **UpnpRegisterRootDevice** (IN const char* DescUrl,
IN Upnp_FunPtr Callback,
IN const void* Cookie,
OUT UpnpDevice_Handle* Hnd)
UpnpRegisterRootDevice *registers a device application with the SDK.* 45

4.3.7 int **UpnpRegisterRootDevice2** (IN Upnp_DescType
descriptionType,
IN const char* description,
IN size_t bufferLen,
IN int config_baseURL,
IN Upnp_FunPtr Fun,
IN const void* Cookie,
OUT UpnpDevice_Handle* Hnd)
UpnpRegisterRootDevice2 *is similar to* **UpnpRegisterRootDevice**, *except that it also allows the description document to be specified as a file or a memory buffer.* 47

4.3.8 int **UpnpUnRegisterClient** (IN UpnpClient_Handle Hnd)
UpnpUnRegisterClient *unregisters a control point application, unsubscribing all active subscriptions.* 49

4.3.9 int **UpnpUnRegisterRootDevice** (IN UpnpDevice_Handle)
Unregisters a root device registered with **UpnpRegisterRootDevice** *or* **UpnpRegisterRootDevice2**. 49

4.3.10 int **UpnpSetContentLength** (IN UpnpClient_Handle Hnd,
IN int contentLength)
Sets the size of the receive buffer for incoming SOAP requests. 50

4.3.1

int **UpnpInit** (IN const char* HostIP, IN unsigned short DestPort)

Initializes the Linux SDK for UPnP Devices.

Initializes the Linux SDK for UPnP Devices. This function must be called before any other API function can be called. It should be called only once. Subsequent calls to this API return a `UPNP_E_INIT` error code.

Optionally, the application can specify a host IP address (in the case of a multi-homed configuration) and a port number to use for all UPnP operations. Since a port number can be used only by one process, multiple processes using the SDK must specify different port numbers.

If unspecified, the SDK will use the first adapter's IP address and an arbitrary port.

This call is synchronous.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to initialize the SDK.
- `UPNP_E_INIT`: The SDK is already initialized.
- `UPNP_E_INIT_FAILED`: The SDK initialization failed for an unknown reason.
- `UPNP_E_SOCKET_BIND`: An error occurred binding a socket.
- `UPNP_E_LISTEN`: An error occurred listening to a socket.
- `UPNP_E_OUTOF_SOCKET`: An error ocurred creating a socket.
- `UPNP_E_INTERNAL_ERROR`: An internal error ocurred.

Parameters:

`HostIP`	The host IP address to use, in string format, for example "192.168.0.1", or `NULL` to use the first adapter's IP address.
`DestPort`	The destination port number to use. 0 will pick an arbitrary free port.

4.3.2

int **UpnpFinish** ()

Terminates the Linux SDK for UPnP Devices.

Terminates the Linux SDK for UPnP Devices. This function must be the last API function called. It should be called only once. Subsequent calls to this API return a `UPNP_E_FINISH` error code.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_FINISH: The SDK is already terminated or it is not initialized.

4.3.3

unsigned short **UpnpGetServerPort** (void)

If '0' is used as the port number in **UpnpInit**, *then this function can be used to retrieve the actual port allocated to the SDK.*

If '0' is used as the port number in **UpnpInit**, then this function can be used to retrieve the actual port allocated to the SDK. If **UpnpInit** has not succeeded then 0 is returned.

Return Value: [unsigned short] The port on which an internal server is listening for UPnP related requests.

4.3.4

char* **UpnpGetServerIpAddress** (void)

If NULL *is used as the IP address in* **UpnpInit**, *then this function can be used to retrieve the actual interface address on which device is running.*

If NULL is used as the IP address in **UpnpInit**, then this function can be used to retrieve the actual interface address on which device is running. If **UpnpInit** has not succeeded then NULL is returned.

Return Value: [char*] The IP address on which an internal server is listening for UPnP related requests.

4.3.5

int **UpnpRegisterClient** (IN Upnp_FunPtr Callback, IN const void* Cookie, OUT UpnpClient_Handle* Hnd)

UpnpRegisterClient *registers a control point application with the SDK.*

UpnpRegisterClient registers a control point application with the SDK. A control point application cannot make any other API calls until it registers using this function.

UpnpRegisterClient is a synchronous call and generates no callbacks. Callbacks can occur as soon as this function returns.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_FINISH`: The SDK is already terminated or is not initialized.
- `UPNP_E_INVALID_PARAM`: Either **Callback** or **Hnd** is not a valid pointer.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to register this control point.

Parameters:

`Callback`	Pointer to a function for receiving asynchronous events.
`Cookie`	Pointer to user data returned with the callback function when invoked.
`Hnd`	Pointer to a variable to store the new control point handle.

4.3.6

int **UpnpRegisterRootDevice** (IN const char* DescUrl, IN Upnp_FunPtr Callback, IN const void* Cookie, OUT UpnpDevice_Handle* Hnd)

UpnpRegisterRootDevice *registers a device application with the SDK.*

UpnpRegisterRootDevice registers a device application with the SDK. A device application cannot make any other API calls until it registers using this function. Device applications can also register as control points (see **UpnpRegisterClient** to get a control point handle to perform control point functionality).

UpnpRegisterRootDevice is synchronous and does not generate any callbacks. Callbacks can occur as soon as this function returns.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_FINISH`: The SDK is already terminated or is not initialized.
- `UPNP_E_INVALID_DESC`: The description document was not a valid device description.
- `UPNP_E_INVALID_URL`: The URL for the description document is not valid.
- `UPNP_E_INVALID_PARAM`: Either **Callback** or **Hnd** is not a valid pointer or **DescURL** is `NULL`.
- `UPNP_E_NETWORK_ERROR`: A network error occurred.
- `UPNP_E_SOCKET_WRITE`: An error or timeout occurred writing to a socket.
- `UPNP_E_SOCKET_READ`: An error or timeout occurred reading from a socket.
- `UPNP_E_SOCKET_BIND`: An error occurred binding a socket.
- `UPNP_E_SOCKET_CONNECT`: An error occurred connecting the socket.
- `UPNP_E_OUTOF_SOCKET`: Too many sockets are currently allocated.
- `UPNP_E_OUTOF_MEMORY`: There are insufficient resources to register this root device.

Parameters:

`DescUrl`	Pointer to a string containing the description URL for this root device instance.
`Callback`	Pointer to the callback function for receiving asynchronous events.
`Cookie`	Pointer to user data returned with the callback function when invoked.
`Hnd`	Pointer to a variable to store the new device handle.

4.3.7

int **UpnpRegisterRootDevice2** (IN Upnp_DescType descriptionType, IN const char* description, IN size_t bufferLen, IN int config_baseURL, IN Upnp_FunPtr Fun, IN const void* Cookie, OUT UpnpDevice_Handle* Hnd)

UpnpRegisterRootDevice2 *is similar to* **UpnpRegisterRootDevice**, *except that it also allows the description document to be specified as a file or a memory buffer.*

UpnpRegisterRootDevice2 is similar to **UpnpRegisterRootDevice**, except that it also allows the description document to be specified as a file or a memory buffer. The description can also be configured to have the correct IP and port address.

NOTE: For the configuration to be functional, the internal web server MUST be present. In addition, the web server MUST be activated (using **UpnpSetWebServerRootDir**) before calling this function. The only condition where the web server can be absent is if the description document is specified as a URL and no configuration is required (i.e. `config_baseURL = 0`.)

UpnpRegisterRootDevice2 is synchronous and does not generate any callbacks. Callbacks can occur as soon as this function returns.

Examples of using different types of description documents:

```
1) Description specified as a URL:
      descriptionType == UPNPREG_URL_DESC
      description is the URL
      bufferLen = 0 (ignored)
2) Description specified as a file:
      descriptionType == UPNPREG_FILENAME_DESC
      description is a filename
      bufferLen = 0 (ignored)
3) Description specified as a memory buffer:
      descriptionType == UPNPREG_BUF_DESC
      description is pointer to a memory buffer
      bufferLen == length of memory buffer
```

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_FINISH: The SDK is already terminated or is not initialized.
- UPNP_E_INVALID_DESC: The description document is not a valid device description.
- UPNP_E_INVALID_PARAM: Either **Callback** or **Hnd** is not a valid pointer or **DescURL** is NULL.
- UPNP_E_NETWORK_ERROR: A network error occurred.
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket.
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket.
- UPNP_E_SOCKET_BIND: An error occurred binding a socket.
- UPNP_E_SOCKET_CONNECT: An error occurred connecting the socket.
- UPNP_E_OUTOF_SOCKET: Too many sockets are currently allocated.
- UPNP_E_OUTOF_MEMORY: There are insufficient resources to register this root device.
- UPNP_E_URL_TOO_BIG: Length of the URL is bigger than the internal buffer.
- UPNP_E_FILE_NOT_FOUND: The description file could not be found.
- UPNP_E_FILE_READ_ERROR: An error occurred reading the description file.
- UPNP_E_INVALID_URL: The URL to the description document is invalid.
- UPNP_E_EXT_NOT_XML: The URL to the description document or file should have a .xml extension.
- UPNP_E_NO_WEB_SERVER: The internal web server has been compiled out; the SDK cannot configure itself from the description document.

Parameters:

`descriptionType`	The type of the description document.
`description`	Treated as a URL, file name or memory buffer depending on description type.
`bufferLen`	The length of memory buffer if passing a description in a buffer, otherwise it is ignored.
`config_baseURL`	If nonzero, `URLBase` of description document is configured and the description is served using the internal web server.
`Fun`	Pointer to the callback function for receiving asynchronous events.
`Cookie`	Pointer to user data returned with the callback function when invoked.
`Hnd`	Pointer to a variable to store the new device handle.

4.3.8

int **UpnpUnRegisterClient** (IN UpnpClient_Handle Hnd)

UpnpUnRegisterClient *unregisters a control point application, unsubscribing all active subscriptions.*

UpnpUnRegisterClient unregisters a control point application, unsubscribing all active subscriptions. After this call, the **UpnpClient_Handle** is no longer valid.

UpnpUnRegisterClient is a synchronous call and generates no callbacks. The SDK generates no more callbacks after this function returns.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid control point handle.

Parameters:

`Hnd`	The handle of the control point instance to unregister.

4.3.9

int **UpnpUnRegisterRootDevice** (IN UpnpDevice_Handle)

Unregisters a root device registered with **UpnpRegisterRootDevice** *or* **UpnpRegisterRootDevice2**.

Unregisters a root device registered with **UpnpRegisterRootDevice** or **UpnpRegisterRootDevice2**. After this call, the **UpnpDevice_Handle** is no longer valid. For all advertisements that have not yet expired, the SDK sends a device unavailable message automatically.

UpnpUnRegisterRootDevice is a synchronous call and generates no callbacks. Once this call returns, the SDK will no longer generate callbacks to the application.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.

Parameters:

`UpnpDevice_Handle`	The handle of the root device instance to unregister.

4.3.10

int **UpnpSetContentLength** (IN UpnpClient_Handle Hnd, IN int contentLength)

Sets the size of the receive buffer for incoming SOAP requests.

Sets the size of the receive buffer for incoming SOAP requests. This API allows devices that have memory constraints to exhibit consistent behaviour if the size of the incoming SOAP request exceeds the memory that device can allocate for incoming SOAP messages. The default value set by the SDK for the buffer is 16K bytes. Trying to set a value greater than 32K will result in an error.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.
- `UPNP_E_LARGE_BUFFER_SIZE`: The buffer size requested was too large.

Parameters:

`Hnd`	The handle of the device instance for which the coincoming content length needs to be set.
`contentLength`	The maximum permissible content length for incoming SOAP actions.

4.4 Discovery

Names

4.4.1 int **UpnpSearchAsync** (IN UpnpClient_Handle Hnd, IN int Mx, IN const char* Target, IN const void* Cookie) **UpnpSearchAsync** *searches for devices matching the given search target.* 51

4.4.2 int **UpnpSendAdvertisement** (IN UpnpDevice_Handle Hnd, IN int Exp) **UpnpSendAdvertisement** *sends out the discovery announcements for all devices and services for a device.* 52

4.4.1

int **UpnpSearchAsync** (IN UpnpClient_Handle Hnd, IN int Mx, IN const char* Target, IN const void* Cookie)

UpnpSearchAsync *searches for devices matching the given search target.*

UpnpSearchAsync searches for devices matching the given search target. The function returns immediately and the SDK calls the default callback function, registered during the **UpnpRegisterClient** call, for each matching root device, device, or service. The application specifies the search type by the **Target** parameter.

Note that there is no way for the SDK to distinguish which client instance issued a particular search. Therefore, the client can get search callbacks that do not match the original criteria of the search. Also, the application will receive multiple callbacks for each search.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid control point handle.
- `UPNP_E_INVALID_PARAM`: **Target** is `NULL`.

Parameters:

`Hnd`	The handle of the client performing the search.
`Mx`	The time, in seconds, to wait for responses. If the time is greater than `MAX_SEARCH_TIME` then the time is set to `MAX_SEARCH_TIME`. If the time is less than `MIN_SEARCH_TIME` then the time is set to `MIN_SEARCH_TIME`.
`Target`	The search target as defined in the UPnP Device Architecture v1.0 specification.
`Cookie`	The user data to pass when the callback function is invoked.

4.4.2

int **UpnpSendAdvertisement** (IN UpnpDevice_Handle Hnd, IN int Exp)

UpnpSendAdvertisement *sends out the discovery announcements for all devices and services for a device.*

UpnpSendAdvertisement sends out the discovery announcements for all devices and services for a device. Each announcement is made with the same expiration time.

UpnpSendAdvertisement is a synchronous call.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.
- `UPNP_E_OUTOF_MEMORY`: There are insufficient resources to send future advertisements.

Parameters:

`Hnd` The device handle for which to send out the announcements.

`Exp` The expiration age, in seconds, of the announcements.

4.5

Control

Names

4.5.1 int **UpnpGetServiceVarStatus** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* VarName, OUT DOMString* StVarVal) **UpnpGetServiceVarStatus** *queries the state of a state variable of a service on another device.* 54

4.5.2 int **UpnpGetServiceVarStatusAsync** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* VarName, IN Upnp_FunPtr Fun, IN const void* Cookie)

UpnpGetServiceVarStatusAsync
queries the state of a variable of a service, generating a callback when the operation is complete. 55

4.5.3 int **UpnpSendAction** (IN UpnpClient_Handle Hnd,
IN const char* ActionURL,
IN const char* ServiceType,
IN const char* DevUDN,
IN IXML_Document* Action,
OUT IXML_Document** RespNode)
UpnpSendAction *sends a message to change a state variable in a service.* 55

4.5.4 int **UpnpSendActionEx** (IN UpnpClient_Handle Hnd,
IN const char* ActionURL,
IN const char* ServiceType,
IN const char* DevUDN,
IN IXML_Document* Header,
IN IXML_Document* Action,
OUT IXML_Document** RespNode)
UpnpSendActionEx *sends a message to change a state variable in a service.* 56

4.5.5 int **UpnpSendActionAsync** (IN UpnpClient_Handle Hnd,
IN const char* ActionURL,
IN const char* ServiceType,
IN const char* DevUDN,
IN IXML_Document* Action,
IN Upnp_FunPtr Fun,
IN const void* Cookie)
UpnpSendActionAsync *sends a message to change a state variable in a service, generating a callback when the operation is complete.* 57

4.5.6 int **UpnpSendActionExAsync** (IN UpnpClient_Handle Hnd,
IN const char* ActionURL,
IN const char* ServiceType,
IN const char* DevUDN,
IN IXML_Document* Header,
IN IXML_Document* Action,
IN Upnp_FunPtr Fun,
IN const void* Cookie)
UpnpSendActionExAsync *sends a message to change a state variable in a service, generating a callback when the operation is complete.* 58

4.5.1

int **UpnpGetServiceVarStatus** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* VarName, OUT DOMString* StVarVal)

UpnpGetServiceVarStatus *queries the state of a state variable of a service on another device.*

UpnpGetServiceVarStatus queries the state of a state variable of a service on another device. This is a synchronous call. A positive return value indicates a SOAP error code, whereas a negative return code indicates an SDK error code. **Note that the use of this function is deprecated by the UPnP Forum**.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid control point handle.
- `UPNP_E_INVALID_URL`: **ActionUrl** is not a valid URL.
- `UPNP_E_INVALID_DESC`: The XML document was not found or it does not contain a valid XML description.
- `UPNP_E_INVALID_PARAM`: **StVarVal** is not a valid pointer or **VarName** or **ActionUrl** is `NULL`.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.
- `UPNP_SOAP_E_INVALID_VAR`: The given variable is invalid according to the device.

Parameters:

`Hnd`	The handle of the control point.
`ActionURL`	The URL of the service.
`VarName`	The name of the variable to query.
`StVarVal`	The pointer to store the value for **VarName**. The SDK allocates this string and the caller needs to free it using **ixmlFreeDOMString**.

4.5.2

int **UpnpGetServiceVarStatusAsync** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* VarName, IN Upnp_FunPtr Fun, IN const void* Cookie)

UpnpGetServiceVarStatusAsync *queries the state of a variable of a service, generating a callback when the operation is complete.*

UpnpGetServiceVarStatusAsync queries the state of a variable of a service, generating a callback when the operation is complete. **Note that the use of this function is deprecated by the UPnP Forum**.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid control point handle.
- `UPNP_E_INVALID_URL`: The **ActionUrl** is not a valid URL.
- `UPNP_E_INVALID_PARAM`: **VarName**, **Fun** or **ActionUrl** is not a valid pointer.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`Hnd`	The handle of the control point.
`ActionURL`	The URL of the service.
`VarName`	The name of the variable to query.
`Fun`	Pointer to a callback function to be invoked when the operation is complete.
`Cookie`	Pointer to user data to pass to the callback function when invoked.

4.5.3

int **UpnpSendAction** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* ServiceType, IN const char* DevUDN, IN IXML_Document* Action, OUT IXML_Document** RespNode)

UpnpSendAction *sends a message to change a state variable in a service.*

UpnpSendAction sends a message to change a state variable in a service. This is a synchronous call that does not return until the action is complete.

Note that a positive return value indicates a SOAP-protocol error code. In this case, the error description can be retrieved from **RespNode**. A negative return value indicates an SDK error.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_HANDLE: The handle is not a valid control point handle.
- UPNP_E_INVALID_URL: **ActionUrl** is not a valid URL.
- UPNP_E_INVALID_ACTION: This action is not valid.
- UPNP_E_INVALID_DEVICE: **DevUDN** is not a valid device.
- UPNP_E_INVALID_PARAM: **ServiceType**, **Action**, **ActionUrl**, or **RespNode** is not a valid pointer.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to complete this operation.

Parameters:

Hnd	The handle of the control point sending the action.
ActionURL	The action URL of the service.
ServiceType	The type of the service.
DevUDN	This parameter is ignored and must beNULL.
Action	The DOM document for the action.
RespNode	The DOM document for the response to the action. The SDK allocates this document and the caller needs to free it.

4.5.4

int **UpnpSendActionEx** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* ServiceType, IN const char* DevUDN, IN IXML_Document* Header, IN IXML_Document* Action, OUT IXML_Document** RespNode)

UpnpSendActionEx *sends a message to change a state variable in a service.*

UpnpSendActionEx sends a message to change a state variable in a service. This is a synchronous call that does not return until the action is complete.

Note that a positive return value indicates a SOAP-protocol error code. In this case, the error description can be retrieved from **RespNode**. A negative return value indicates an SDK error.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_HANDLE: The handle is not a valid control point handle.
- UPNP_E_INVALID_URL: **ActionUrl** is not a valid URL.
- UPNP_E_INVALID_ACTION: This action is not valid.
- UPNP_E_INVALID_DEVICE: **DevUDN** is not a valid device.
- UPNP_E_INVALID_PARAM: **ServiceType**, **Action**, **ActionUrl**, or **RespNode** is not a valid pointer.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to complete this operation.

Parameters:

Hnd	The handle of the control point sending the action.
ActionURL	The action URL of the service.
ServiceType	The type of the service.
DevUDN	This parameter is ignored and must beNULL.
Header	The DOM document for the SOAP header. This may be NULL if the header is not required.
Action	The DOM document for the action.
RespNode	The DOM document for the response to the action. The SDK allocates this document and the caller needs to free it.

4.5.5

int **UpnpSendActionAsync** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* ServiceType, IN const char* DevUDN, IN IXML_Document* Action, IN Upnp_FunPtr Fun, IN const void* Cookie)

UpnpSendActionAsync *sends a message to change a state variable in a service, generating a callback when the operation is complete.*

UpnpSendActionAsync sends a message to change a state variable in a service, generating a callback when the operation is complete. See **UpnpSendAction** for comments on positive return values. These positive return values are sent in the event struct associated with the UPNP_CONTROL_ACTION_COMPLETE event.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid control point handle.
- `UPNP_E_INVALID_URL`: **ActionUrl** is an invalid URL.
- `UPNP_E_INVALID_DEVICE`: **DevUDN** is an invalid device.
- `UPNP_E_INVALID_PARAM`: Either **Fun** is not a valid callback function or **ServiceType**, **Act**, or **ActionUrl** is `NULL`.
- `UPNP_E_INVALID_ACTION`: This action is not valid.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`Hnd`	The handle of the control point sending the action.
`ActionURL`	The action URL of the service.
`ServiceType`	The type of the service.
`DevUDN`	This parameter is ignored and must be`NULL`.
`Action`	The DOM document for the action to perform on this device.
`Fun`	Pointer to a callback function to be invoked when the operation completes.
`Cookie`	Pointer to user data that to be passed to the callback when invoked.

4.5.6

int **UpnpSendActionExAsync** (IN UpnpClient_Handle Hnd, IN const char* ActionURL, IN const char* ServiceType, IN const char* DevUDN, IN IXML_Document* Header, IN IXML_Document* Action, IN Upnp_FunPtr Fun, IN const void* Cookie)

UpnpSendActionExAsync *sends a message to change a state variable in a service, generating a callback when the operation is complete.*

UpnpSendActionExAsync sends a message to change a state variable in a service, generat-

ing a callback when the operation is complete. See **UpnpSendAction** for comments on positive return values. These positive return values are sent in the event struct associated with the UPNP_CONTROL_ACTION_COMPLETE event.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_HANDLE: The handle is not a valid control point handle.
- UPNP_E_INVALID_URL: **ActionUrl** is an invalid URL.
- UPNP_E_INVALID_DEVICE: **DevUDN** is an invalid device.
- UPNP_E_INVALID_PARAM: Either **Fun** is not a valid callback function or **ServiceType**, **Act**, or **ActionUrl** is NULL.
- UPNP_E_INVALID_ACTION: This action is not valid.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to complete this operation.

Parameters:

Hnd	The handle of the control point sending the action.
ActionURL	The action URL of the service.
ServiceType	The type of the service.
DevUDN	This parameter is ignored and must beNULL.
Header	The DOM document for the SOAP header. This may be NULL if the header is not required.
Action	The DOM document for the action to perform on this device.
Fun	Pointer to a callback function to be invoked when the operation completes.
Cookie	Pointer to user data that to be passed to the callback when invoked.

4.6 Eventing

Names

4.6.1 int **UpnpAcceptSubscription** (IN UpnpDevice_Handle Hnd,
IN const char* DevID,
IN const char* ServID,
IN const char** VarName,
IN const char** NewVal,
IN int cVariables,
IN Upnp_SID SubsId)

UpnpAcceptSubscription *accepts a subscription request and sends out the current state of the eventable variables for a service.* 62

4.6.2 int **UpnpAcceptSubscriptionExt** (IN UpnpDevice_Handle Hnd,
IN const char* DevID,
IN const char* ServID,
IN IXML_Document* PropSet,
IN Upnp_SID SubsId)
UpnpAcceptSubscriptionExt *is similar to* **UpnpAcceptSubscription** *except that it takes a DOM document for the variables to event rather than an array of strings.* 63

4.6.3 int **UpnpNotify** (IN UpnpDevice_Handle, IN const char* DevID,
IN const char* ServID,
IN const char** VarName,
IN const char** NewVal, IN int cVariables)
UpnpNotify *sends out an event change notification to all control points subscribed to a particular service.* 64

4.6.4 int **UpnpNotifyExt** (IN UpnpDevice_Handle,
IN const char* DevID,
IN const char* ServID,
IN IXML_Document* PropSet)
UpnpNotifyExt *is similar to* **UpnpNotify** *except that it takes a DOM document for the event rather than an array of strings.* 65

4.6.5 int **UpnpRenewSubscription** (IN UpnpClient_Handle Hnd,
INOUT int* TimeOut,
IN Upnp_SID SubsId)
UpnpRenewSubscription *renews a subscription that is about to expire.* 65

4.6.6 int **UpnpRenewSubscriptionAsync** (IN UpnpClient_Handle
Hnd, IN int TimeOut,
IN Upnp_SID SubsId,
IN Upnp_FunPtr Fun,
IN const void* Cookie)
UpnpRenewSubscriptionAsync *renews a subscription that is about to expire, generating a callback when the operation is complete.* 67

4.6.7 int **UpnpSetMaxSubscriptions** (IN UpnpDevice_Handle Hnd,
IN int MaxSubscriptions)

UpnpSetMaxSubscriptions *sets the maximum number of subscriptions accepted per service.* 69

4.6.8 int **UpnpSetMaxSubscriptionTimeOut** (IN UpnpDevice_Handle Hnd, IN int MaxSubscriptionTimeOut)
UpnpSetMaxSubscriptionTimeOut *sets the maximum time-out accepted for a subscription request or renewal.* 70

4.6.9 int **UpnpSubscribe** (IN UpnpClient_Handle Hnd, IN const char* PublisherUrl, INOUT int* TimeOut, OUT Upnp_SID SubsId)
UpnpSubscribe *registers a control point to receive event notifications from another device.* 70

4.6.10 int **UpnpSubscribeAsync** (IN UpnpClient_Handle Hnd, IN const char* PublisherUrl, IN int TimeOut, IN Upnp_FunPtr Fun, IN const void* Cookie)
UpnpSubscribeAsync *performs the same operation as* **UpnpSubscribe**, *but returns immediately and calls the registered callback function when the operation is complete.* 72

4.6.11 int **UpnpUnSubscribe** (IN UpnpClient_Handle Hnd, IN Upnp_SID SubsId)
UpnpUnSubscribe *removes the subscription of a control point from a service previously subscribed to using* **UpnpSubscribe** *or* **UpnpSubscribeAsync**. 74

4.6.12 int **UpnpUnSubscribeAsync** (IN UpnpClient_Handle Hnd, IN Upnp_SID SubsId, IN Upnp_FunPtr Fun, IN const void* Cookie)
UpnpUnSubscribeAsync *removes a subscription of a control point from a service previously subscribed to using* **UpnpSubscribe** *or* **UpnpSubscribeAsync**, *generating a callback when the operation is complete.* 75

4.6.1

int **UpnpAcceptSubscription** (IN UpnpDevice_Handle Hnd, IN const char* DevID, IN const char* ServID, IN const char** VarName, IN const char** NewVal, IN int cVariables, IN Upnp_SID SubsId)

UpnpAcceptSubscription *accepts a subscription request and sends out the current state of the eventable variables for a service.*

UpnpAcceptSubscription accepts a subscription request and sends out the current state of the eventable variables for a service. The device application should call this function when it receives a `UPNP_EVENT_SUBSCRIPTION_REQUEST` callback. This function is synchronous and generates no callbacks.

UpnpAcceptSubscription can be called during the execution of a callback function.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.
- `UPNP_E_INVALID_SERVICE`: The **DevId/ServId** pair refers to an invalid service.
- `UPNP_E_INVALID_SID`: The specified subscription ID is not valid.
- `UPNP_E_INVALID_PARAM`: Either **VarName**, **NewVal**, **DevID**, or **ServID** is not a valid pointer or **cVariables** is less than zero.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`Hnd`	The handle of the device.
`DevID`	The device ID of the subdevice of the service generating the event.
`ServID`	The unique service identifier of the service generating the event.
`VarName`	Pointer to an array of event variables.
`NewVal`	Pointer to an array of values for the event variables.
`cVariables`	The number of event variables in **VarName**.
`SubsId`	The subscription ID of the newly registered control point.

4.6.2

int **UpnpAcceptSubscriptionExt** (IN UpnpDevice_Handle Hnd, IN const char* DevID, IN const char* ServID, IN IXML_Document* PropSet, IN Upnp_SID SubsId)

UpnpAcceptSubscriptionExt *is similar to* **UpnpAcceptSubscription** *except that it takes a DOM document for the variables to event rather than an array of strings.*

UpnpAcceptSubscriptionExt is similar to **UpnpAcceptSubscription** except that it takes a DOM document for the variables to event rather than an array of strings. This function is sychronous and generates no callbacks.

UpnpAcceptSubscriptionExt can be called during the execution of a callback function.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.
- `UPNP_E_INVALID_SERVICE`: The **DevId/ServId** pair refers to an invalid service.
- `UPNP_E_INVALID_SID`: The specified subscription ID is not valid.
- `UPNP_E_INVALID_PARAM`: Either **VarName**, **NewVal**, **DevID**, **ServID**, or **PropSet** is not a valid pointer.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`Hnd`	The handle of the device.
`DevID`	The device ID of the subdevice of the service generating the event.
`ServID`	The unique service identifier of the service generating the event.
`PropSet`	The DOM document for the property set. Property set documents must conform tothe XML schema defined in section 4.3 of theUniversal Plug and Play Device Architecturespecification.
`SubsId`	The subscription ID of the newly registered control point.

4.6.3

int **UpnpNotify** (IN UpnpDevice_Handle, IN const char* DevID, IN const char* ServID, IN const char** VarName, IN const char** NewVal, IN int cVariables)

UpnpNotify *sends out an event change notification to all control points subscribed to a particular service.*

UpnpNotify sends out an event change notification to all control points subscribed to a particular service. This function is synchronous and generates no callbacks.

UpnpNotify may be called during a callback function to send out a notification.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.
- `UPNP_E_INVALID_SERVICE`: The **DevId**/**ServId** pair refers to an invalid service.
- `UPNP_E_INVALID_PARAM`: Either **VarName**, **NewVal**, **DevID**, or **ServID** is not a valid pointer or **cVariables** is less than zero.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`UpnpDevice_Handle`	The handle to the device sending the event.
`DevID`	The device ID of the subdevice of the service generating the event.
`ServID`	The unique identifier of the service generating the event.
`VarName`	Pointer to an array of variables that have changed.
`NewVal`	Pointer to an array of new values for those variables.
`cVariables`	The count of variables included in this notification.

4.6.4

int **UpnpNotifyExt** (IN UpnpDevice_Handle, IN const char* DevID, IN const char* ServID, IN IXML_Document* PropSet)

UpnpNotifyExt *is similar to* **UpnpNotify** *except that it takes a DOM document for the event rather than an array of strings.*

UpnpNotifyExt is similar to **UpnpNotify** except that it takes a DOM document for the event rather than an array of strings. This function is synchronous and generates no callbacks.

UpnpNotifyExt may be called during a callback function to send out a notification.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.
- `UPNP_E_INVALID_SERVICE`: The **DevId/ServId** pair refers to an invalid service.
- `UPNP_E_INVALID_PARAM`: Either **VarName**, **NewVal**, **DevID**, **ServID**, or **PropSet** is not a valid pointer or **cVariables** is less than zero.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`UpnpDevice_Handle`	The handle to the device sending the event.
`DevID`	The device ID of the subdevice of the service generating the event.
`ServID`	The unique identifier of the service generating the event.
`PropSet`	The DOM document for the property set. Property set documents must conform to the XML schema defined in section 4.3 of the Universal Plug and Play Device Architecture specification.

4.6.5

int **UpnpRenewSubscription** (IN UpnpClient_Handle Hnd, INOUT int* TimeOut, IN Upnp_SID SubsId)

UpnpRenewSubscription *renews a subscription that is about to expire.*

UpnpRenewSubscription renews a subscription that is about to expire. This function is synchronous.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_HANDLE: The handle is not a valid control point handle.
- UPNP_E_INVALID_PARAM: **Timeout** is not a valid pointer.
- UPNP_E_INVALID_SID: The SID being passed to this function is not a valid subscription ID.
- UPNP_E_NETWORK_ERROR: A network error occured.
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket.
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket.
- UPNP_E_SOCKET_BIND: An error occurred binding a socket.
- UPNP_E_SOCKET_CONNECT: An error occurred connecting to **PublisherUrl**.
- UPNP_E_OUTOF_SOCKET: An error occurred creating a socket.
- UPNP_E_BAD_RESPONSE: An error occurred in response from the publisher.
- UPNP_E_SUBSCRIBE_UNACCEPTED: The publisher refused the subscription renew.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to complete this operation.

Parameters:

Hnd	The handle of the control point that is renewing the subscription.
TimeOut	Pointer to a variable containing the requested subscription time. Upon return, it contains the actual renewal time.
SubsId	The ID for the subscription to renew.

4.6.6

int **UpnpRenewSubscriptionAsync** (IN UpnpClient_Handle Hnd, IN int TimeOut, IN Upnp_SID SubsId, IN Upnp_FunPtr Fun, IN const void* Cookie)

UpnpRenewSubscriptionAsync *renews a subscription that is about to expire, generating a callback when the operation is complete.*

UpnpRenewSubscriptionAsync renews a subscription that is about to expire, generating a callback when the operation is complete.

Note that many of the error codes for this function are returned in the **Upnp_Event_Subscribe** structure. In those cases, the function returns `UPNP_E_SUCCESS` and the appropriate error code will be in the **Upnp_Event_Subscribe.ErrCode** field in the **Event** structure passed to the callback.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_HANDLE: The handle is not a valid control point handle.
- UPNP_E_INVALID_SID: The **SubsId** is not a valid subscription ID.
- UPNP_E_INVALID_PARAM: Either **Fun** is not a valid callback function pointer or **Timeout** is less than zero but is not UPNP_INFINITE.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to complete this operation.
- UPNP_E_NETWORK_ERROR: A network error occured (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_BIND: An error occurred binding the socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_CONNECT: An error occurred connecting to **PublisherUrl** (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_OUTOF_SOCKET: An error occurred creating socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_BAD_RESPONSE: An error occurred in response from the publisher (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SUBSCRIBE_UNACCEPTED: The publisher refused the subscription request (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).

Parameters:

`Hnd`	The handle of the control point that is renewing the subscription.
`TimeOut`	The requested subscription time. The actual timeout value is returned when the callback function is called.
`SubsId`	The ID for the subscription to renew.
`Fun`	Pointer to a callback function to be invoked when the renewal is complete.
`Cookie`	Pointer to user data passed to the callback function when invoked.

4.6.7

int **UpnpSetMaxSubscriptions** (IN UpnpDevice_Handle Hnd, IN int MaxSubscriptions)

UpnpSetMaxSubscriptions *sets the maximum number of subscriptions accepted per service.*

UpnpSetMaxSubscriptions sets the maximum number of subscriptions accepted per service. The default value accepts as many as system resources allow. If the number of current subscriptions for a service is greater than the requested value, the SDK accepts no new subscriptions or renewals, however, the SDK does not remove any current subscriptions.

Return Value:

`[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.

Parameters:

`Hnd`	The handle of the device for which the maximum number of subscriptions is being set.
`MaxSubscriptions`	The maximum number of subscriptions to be allowed per service.

4.6.8

int **UpnpSetMaxSubscriptionTimeOut** (IN UpnpDevice_Handle Hnd, IN int MaxSubscriptionTimeOut)

UpnpSetMaxSubscriptionTimeOut *sets the maximum time-out accepted for a subscription request or renewal.*

UpnpSetMaxSubscriptionTimeOut sets the maximum time-out accepted for a subscription request or renewal. The default value accepts the time-out set by the control point. If a control point requests a subscription time-out less than or equal to the maximum, the SDK grants the value requested by the control point. If the time-out is greater, the SDK returns the maximum value.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid device handle.

Parameters:

`Hnd`	The handle of the device for which the maximum subscription time-out is being set.
`MaxSubscriptionTimeOut`	The maximum subscription time-out to be accepted.

4.6.9

int **UpnpSubscribe** (IN UpnpClient_Handle Hnd, IN const char* PublisherUrl, INOUT int* TimeOut, OUT Upnp_SID SubsId)

UpnpSubscribe *registers a control point to receive event notifications from another device.*

UpnpSubscribe registers a control point to receive event notifications from another device. This operation is synchronous.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_HANDLE: The handle is not a valid control point handle.
- UPNP_E_INVALID_URL: **PublisherUrl** is not a valid URL.
- UPNP_E_INVALID_PARAM: **Timeout** is not a valid pointer or **SubsId** or **PublisherUrl** is NULL.
- UPNP_E_NETWORK_ERROR: A network error occured.
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket.
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket.
- UPNP_E_SOCKET_BIND: An error occurred binding a socket.
- UPNP_E_SOCKET_CONNECT: An error occurred connecting to **PublisherUrl**.
- UPNP_E_OUTOF_SOCKET: An error occurred creating a socket.
- UPNP_E_BAD_RESPONSE: An error occurred in response from the publisher.
- UPNP_E_SUBSCRIBE_UNACCEPTED: The publisher refused the subscription request.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to complete this operation.

Parameters:

Hnd	The handle of the control point.
PublisherUrl	The URL of the service to subscribe to.
TimeOut	Pointer to a variable containing the requested subscription time. Upon return, it contains the actual subscription time returned from the service.
SubsId	Pointer to a variable to receive the subscription ID (SID).

4.6.10

int **UpnpSubscribeAsync** (IN UpnpClient_Handle Hnd, IN const char* PublisherUrl, IN int TimeOut, IN Upnp_FunPtr Fun, IN const void* Cookie)

UpnpSubscribeAsync *performs the same operation as* **UpnpSubscribe**, *but returns immediately and calls the registered callback function when the operation is complete.*

UpnpSubscribeAsync performs the same operation as **UpnpSubscribe**, but returns immediately and calls the registered callback function when the operation is complete.

Note that many of the error codes for this function are returned in the **Upnp_Event_Subscribe** structure. In those cases, the function returns `UPNP_E_SUCCESS` and the appropriate error code will be in the **Upnp_Event_Subscribe.ErrCode** field in the **Event** structure passed to the callback.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_HANDLE: The handle is not a valid control point handle.
- UPNP_E_INVALID_URL: The **PublisherUrl** is not a valid URL.
- UPNP_E_INVALID_PARAM: Either **TimeOut** or **Fun** or **PublisherUrl** is not a valid pointer.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to complete this operation.
- UPNP_E_NETWORK_ERROR: A network error occured (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_BIND: An error occurred binding the socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SOCKET_CONNECT: An error occurred connecting to **PublisherUrl** (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_OUTOF_SOCKET: An error occurred creating the socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_BAD_RESPONSE: An error occurred in response from the publisher (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- UPNP_E_SUBSCRIBE_UNACCEPTED: The publisher refused the subscription request (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).

Parameters:		
	`Hnd`	The handle of the control point that is subscribing.
	`PublisherUrl`	The URL of the service to subscribe to.
	`TimeOut`	The requested subscription time. Upon return, it contains the actual subscription time returned from the service.
	`Fun`	Pointer to the callback function for this subscribe request.
	`Cookie`	A user data value passed to the callback function when invoked.

4.6.11

int **UpnpUnSubscribe** (IN UpnpClient_Handle Hnd, IN Upnp_SID SubsId)

UpnpUnSubscribe *removes the subscription of a control point from a service previously subscribed to using* **UpnpSubscribe** *or* **UpnpSubscribeAsync**.

UpnpUnSubscribe removes the subscription of a control point from a service previously subscribed to using **UpnpSubscribe** or **UpnpSubscribeAsync**. This is a synchronous call.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid control point handle.
- `UPNP_E_INVALID_SID`: The **SubsId** is not a valid subscription ID.
- `UPNP_E_NETWORK_ERROR`: A network error occured.
- `UPNP_E_SOCKET_WRITE`: An error or timeout occurred writing to a socket.
- `UPNP_E_SOCKET_READ`: An error or timeout occurred reading from a socket.
- `UPNP_E_SOCKET_BIND`: An error occurred binding a socket.
- `UPNP_E_SOCKET_CONNECT`: An error occurred connecting to **PublisherUrl**.
- `UPNP_E_OUTOF_SOCKET`: An error ocurred creating a socket.
- `UPNP_E_BAD_RESPONSE`: An error occurred in response from the publisher.
- `UPNP_E_UNSUBSCRIBE_UNACCEPTED`: The publisher refused the unsubscribe request (the client is still unsubscribed and no longer receives events).
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`Hnd`	The handle of the subscribed control point.
`SubsId`	The ID returned when the control point subscribed to the service.

4.6.12

int **UpnpUnSubscribeAsync** (IN UpnpClient_Handle Hnd, IN Upnp_SID SubsId, IN Upnp_FunPtr Fun, IN const void* Cookie)

UpnpUnSubscribeAsync *removes a subscription of a control point from a service previously subscribed to using* **UpnpSubscribe** *or* **UpnpSubscribeAsync***, generating a callback when the operation is complete.*

UpnpUnSubscribeAsync removes a subscription of a control point from a service previously subscribed to using **UpnpSubscribe** or **UpnpSubscribeAsync**, generating a callback when the operation is complete.

Note that many of the error codes for this function are returned in the **Upnp_Event_Subscribe** structure. In those cases, the function returns `UPNP_E_SUCCESS` and the appropriate error code will be in the **Upnp_Event_Subscribe.ErrCode** field in the **Event** structure passed to the callback.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_HANDLE`: The handle is not a valid control point handle.
- `UPNP_E_INVALID_SID`: The **SubsId** is not a valid SID.
- `UPNP_E_INVALID_PARAM`: **Fun** is not a valid callback function pointer.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.
- `UPNP_E_NETWORK_ERROR`: A network error occured (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- `UPNP_E_SOCKET_WRITE`: An error or timeout occurred writing to a socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- `UPNP_E_SOCKET_READ`: An error or timeout occurred reading from a socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- `UPNP_E_SOCKET_BIND`: An error occurred binding the socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- `UPNP_E_SOCKET_CONNECT`: An error occurred connecting to **PublisherUrl** (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- `UPNP_E_OUTOF_SOCKET`: An error occurred creating a socket (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- `UPNP_E_BAD_RESPONSE`: An error occurred in response from the publisher (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).
- `UPNP_E_UNSUBSCRIBE_UNACCEPTED`: The publisher refused the subscription request (returned in the **Upnp_Event_Subscribe.ErrCode** field as part of the callback).

Parameters:

`Hnd`	The handle of the subscribed control point.
`SubsId`	The ID returned when the control point subscribed to the service.
`Fun`	Pointer to a callback function to be called when the operation is complete.
`Cookie`	Pointer to user data to pass to the callback function when invoked.

4.7 Control Point HTTP API

Names

4.7.1 int **UpnpDownloadUrlItem** (IN const char* url,
OUT char** outBuf,
OUT char* contentType)
UpnpDownloadUrlItem *downloads a file specified in a URL.* 78

4.7.2 int **UpnpOpenHttpGet** (IN const char* url,
IN OUT void** handle,
IN OUT char** contentType,
IN OUT int* contentLength,
IN OUT int* httpStatus,
IN int timeout)
UpnpOpenHttpGet *gets a file specified in a URL.* 79

4.7.3 int **UpnpOpenHttpGetEx** (IN const char* url,
IN OUT void** handle,
IN OUT char** contentType,
IN OUT int* contentLength,
IN OUT int* httpStatus,
IN int lowRange, IN int highRange,
IN int timeout)
UpnpOpenHttpGetEx *gets specified number of bytes from a file specified in the URL.* 80

4.7.4 int **UpnpReadHttpGet** (IN void* handle, IN OUT char* buf,
IN OUT unsigned int* size,
IN int timeout)
UpnpReadHttpGet *gets specified number of bytes from a file specified in a URL.* 82

4.7.5 int **UpnpCloseHttpGet** (IN void* handle)

UpnpCloseHttpGet *closes the connection and frees memory that was allocated for the* **handle** *parameter.* 82

4.7.6 int **UpnpOpenHttpPost** (IN const char* url,
IN OUT void** handle,
IN const char* contentType,
IN int contentLength, IN int timeout)
UpnpOpenHttpPost *makes an HTTP POST request message, opens a connection to the server and sends the POST request to the server if the connection to the server succeeds.* 83

4.7.7 int **UpnpWriteHttpPost** (IN void* handle, IN char* buf,
IN unsigned int* size, IN int timeout)
UpnpWriteHttpPost *sends a request to a server to copy the contents of a buffer to the URI specified in the* **UpnpOpenHttpPost** *call.* 84

4.7.8 int **UpnpCloseHttpPost** (IN void* handle,
IN OUT int* httpStatus,
IN int timeout)
UpnpCloseHttpPost *sends and receives any pending data, closes the connection with the server, and frees memory allocated during the call.* 84

4.7.9 int **UpnpDownloadXmlDoc** (IN const char* url,
OUT IXML_Document** xmlDoc)
UpnpDownloadXmlDoc *downloads an XML document specified in a URL.* 85

4.7.1

int **UpnpDownloadUrlItem** (IN const char* url, OUT char** outBuf,
OUT char* contentType)

UpnpDownloadUrlItem *downloads a file specified in a URL.*

UpnpDownloadUrlItem downloads a file specified in a URL. The SDK allocates the memory for **outBuf** and the application is responsible for freeing this memory. Note that the item is passed as a single buffer. Large items should not be transferred using this function.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_PARAM: Either **url**, **outBuf** or **contentType** is not a valid pointer.
- UPNP_E_INVALID_URL: The **url** is not a valid URL.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to download this file.
- UPNP_E_NETWORK_ERROR: A network error occurred.
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket.
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket.
- UPNP_E_SOCKET_BIND: An error occurred binding a socket.
- UPNP_E_SOCKET_CONNECT: An error occurred connecting a socket.
- UPNP_E_OUTOF_SOCKET: Too many sockets are currently allocated.

Parameters:

url	URL of an item to download.
outBuf	Buffer to store the downloaded item.
contentType	HTTP header value content type if present. It should be at least LINE_SIZE bytes in size.

4.7.2

int **UpnpOpenHttpGet** (IN const char* url, IN OUT void** handle, IN OUT char** contentType, IN OUT int* contentLength, IN OUT int* httpStatus, IN int timeout)

UpnpOpenHttpGet *gets a file specified in a URL.*

UpnpOpenHttpGet gets a file specified in a URL. The SDK allocates the memory for **handle** and **contentType**, the application is responsible for freeing this memory.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_PARAM: Either **url**, **handle**, **contentType**, **contentLength** or **httpStatus** is not a valid pointer.
- UPNP_E_INVALID_URL: The **url** is not a valid URL.
- UPNP_E_OUTOF_MEMORY: Insufficient resources exist to download this file.
- UPNP_E_NETWORK_ERROR: A network error occurred.
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket.
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket.
- UPNP_E_SOCKET_BIND: An error occurred binding a socket.
- UPNP_E_SOCKET_CONNECT: An error occurred connecting a socket.
- UPNP_E_OUTOF_SOCKET: Too many sockets are currently allocated.
- UPNP_E_BAD_RESPONSE: A bad response was received from the remote server.

Parameters:

url	The URL of an item to get.
handle	A pointer to store the handle for this connection.
contentType	A buffer to store the media type of the item.
contentLength	A pointer to store the length of the item.
httpStatus	The status returned on receiving a response message.
timeout	The time out value sent with the request during which a response is expected from the server, failing which, an error is reported back to the user.

4.7.3

int **UpnpOpenHttpGetEx** (IN const char* url, IN OUT void** handle, IN OUT char** contentType, IN OUT int* contentLength, IN OUT int* httpStatus, IN int lowRange, IN int highRange, IN int timeout)

UpnpOpenHttpGetEx *gets specified number of bytes from a file specified in the URL.*

UpnpOpenHttpGetEx gets specified number of bytes from a file specified in the URL. The number of bytes is specified through a low count and a high count which are passed as a range of bytes for the request. The SDK allocates the memory for **handle** and **contentType**, the application is responsible for freeing this memory.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: Either **url**, **handle**, **contentType**, **contentLength** or **httpStatus** is not a valid pointer.
- `UPNP_E_INVALID_URL`: The **url** is not a valid URL.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to download this file.
- `UPNP_E_NETWORK_ERROR`: A network error occurred.
- `UPNP_E_SOCKET_WRITE`: An error or timeout occurred writing to a socket.
- `UPNP_E_SOCKET_READ`: An error or timeout occurred reading from a socket.
- `UPNP_E_SOCKET_BIND`: An error occurred binding a socket.
- `UPNP_E_SOCKET_CONNECT`: An error occurred connecting a socket.
- `UPNP_E_OUTOF_SOCKET`: Too many sockets are currently allocated.
- `UPNP_E_BAD_RESPONSE`: A bad response was received from the remote server.

Parameters:

`url`	The URL of the item to get.
`handle`	A pointer to store the handle for this connection.
`contentType`	A buffer to store the media type of the item.
`contentLength`	A buffer to store the length of the item.
`httpStatus`	The status returned on receiving a response message from the remote server.
`lowRange`	An integer value representing the low end of a range to retrieve.
`highRange`	An integer value representing the high end of a range to retrieve.
`timeout`	A time out value sent with the request during which a response is expected from the server, failing which, an error is reported back to the user.

4.7.4

int **UpnpReadHttpGet** (IN void* handle, IN OUT char* buf, IN OUT unsigned int* size, IN int timeout)

UpnpReadHttpGet *gets specified number of bytes from a file specified in a URL.*

UpnpReadHttpGet gets specified number of bytes from a file specified in a URL.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: Either **handle**, **buf** or **size** is not a valid pointer.
- `UPNP_E_BAD_RESPONSE`: A bad response was received from the remote server.
- `UPNP_E_BAD_HTTPMSG`: Either the request or response was in the incorrect format.

Note: In case of return values, the status code parameter of the passed in handle value may provide additional information on the return value.

Parameters:

`handle`	The token created by the call to **UpnpOpenHttpGet**.
`buf`	The buffer to store the read item.
`size`	The size of the buffer to be read.
`timeout`	The time out value sent with the request during which a response is expected from the server, failing which, an error is reported back to the user.

4.7.5

int **UpnpCloseHttpGet** (IN void* handle)

UpnpCloseHttpGet *closes the connection and frees memory that was allocated for the* **handle** *parameter.*

UpnpCloseHttpGet closes the connection and frees memory that was allocated for the **handle** parameter.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: **handle** is not a valid pointer.

4.7.6

int **UpnpOpenHttpPost** (IN const char* url, IN OUT void** handle, IN const char* contentType, IN int contentLength, IN int timeout)

UpnpOpenHttpPost *makes an HTTP POST request message, opens a connection to the server and sends the POST request to the server if the connection to the server succeeds.*

UpnpOpenHttpPost makes an HTTP POST request message, opens a connection to the server and sends the POST request to the server if the connection to the server succeeds. The SDK allocates the memory for **handle** and **contentType**, the application is responsible for freeing this memory.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: Either **url**, **handle** or **contentType** is not a valid pointer.
- `UPNP_E_INVALID_URL`: The **url** is not a valid URL.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to download this file.
- `UPNP_E_SOCKET_ERROR`: Error occured allocating a socket and resources or an error occurred binding a socket.
- `UPNP_E_SOCKET_WRITE`: An error or timeout occurred writing to a socket.
- `UPNP_E_SOCKET_CONNECT`: An error occurred connecting a socket.
- `UPNP_E_OUTOF_SOCKET`: Too many sockets are currently allocated.

Parameters:

`url`	The URL in which to send the POST request.
`handle`	A pointer in which to store the handle for this connection. This handle is required for futher operations over this connection.
`contentType`	A buffer to store the media type of content being sent.
`contentLength`	The length of the content, in bytes, being posted.
`timeout`	The time out value sent with the request during which a response is expected from the receiver, failing which, an error is reported.

4.7.7

int **UpnpWriteHttpPost** (IN void* handle, IN char* buf, IN unsigned int* size, IN int timeout)

UpnpWriteHttpPost *sends a request to a server to copy the contents of a buffer to the URI specified in the* **UpnpOpenHttpPost** *call.*

UpnpWriteHttpPost sends a request to a server to copy the contents of a buffer to the URI specified in the **UpnpOpenHttpPost** call.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: Either **handle**, **buf** or **size** is not a valid pointer.
- `UPNP_E_SOCKET_WRITE`: An error or timeout occurred writing to a socket.
- `UPNP_E_OUTOF_SOCKET`: Too many sockets are currently allocated.

Parameters:

`handle`	The handle of the connection created by the call to **UpnpOpenHttpPost**.
`buf`	The buffer to be posted.
`size`	The size, in bytes of **buf**.
`timeout`	A timeout value sent with the request during which a response is expected from the server, failing which, an error is reported.

4.7.8

int **UpnpCloseHttpPost** (IN void* handle, IN OUT int* httpStatus, IN int timeout)

UpnpCloseHttpPost *sends and receives any pending data, closes the connection with the server, and frees memory allocated during the call.*

UpnpCloseHttpPost sends and receives any pending data, closes the connection with the server, and frees memory allocated during the call.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_PARAM: Either **handle**, or **httpStatus** is not a valid pointer.
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket.
- UPNP_E_OUTOF_SOCKET: Too many sockets are currently allocated.

Parameters:

handle	The handle of the connection to close, created by the call to **UpnpOpenHttpPost**.
httpStatus	A pointer to a buffer to store the final status of the connection.
timeout	A time out value sent with the request during which a response is expected from the server, failing which, an error is reported.

4.7.9

int **UpnpDownloadXmlDoc** (IN const char* url, OUT IXML_Document** xmlDoc)

UpnpDownloadXmlDoc *downloads an XML document specified in a URL.*

UpnpDownloadXmlDoc downloads an XML document specified in a URL. The SDK parses the document and returns it in the form of a DOM document. The application is responsible for freeing the DOM document.

Return Value: [int] An integer representing one of the following:

- UPNP_E_SUCCESS: The operation completed successfully.
- UPNP_E_INVALID_PARAM: Either **url** or **xmlDoc** is not a valid pointer.
- UPNP_E_INVALID_DESC: The XML document was not found or it does not contain a valid XML description.
- UPNP_E_INVALID_URL: The **url** is not a valid URL.
- UPNP_E_OUTOF_MEMORY: There are insufficient resources to download the XML document.
- UPNP_E_NETWORK_ERROR: A network error occurred.
- UPNP_E_SOCKET_WRITE: An error or timeout occurred writing to a socket.
- UPNP_E_SOCKET_READ: An error or timeout occurred reading from a socket.
- UPNP_E_SOCKET_BIND: An error occurred binding a socket.
- UPNP_E_SOCKET_CONNECT: An error occurred connecting the socket.
- UPNP_E_OUTOF_SOCKET: Too many sockets are currently allocated.

Parameters:

url	URL of the XML document.
xmlDoc	A pointer in which to store the XML document.

4.8 Web Server API

Names

4.8.1 int **UpnpSetWebServerRootDir** (IN const char* rootDir)
UpnpSetWebServerRootDir *sets the document root directory for the internal web server.* 87

4.8.2 int **UpnpSetVirtualDirCallbacks** (IN struct UpnpVirtualDirCallbacks* callbacks)
UpnpSetVirtualDirCallbacks *sets the callback function to be used to access a virtual directory.* 88

4.8.3 int **UpnpEnableWebserver** (IN int enable)

UpnpEnableWebServer *enables or disables the webserver.* 88

4.8.4 int **UpnpIsWebserverEnabled** ()
UpnpIsWebServerEnabled *returns* `TRUE` *if the webserver is enabled, or* `FALSE` *if it is not.* 88

4.8.5 int **UpnpAddVirtualDir** (IN const char* dirName)
UpnpAddVirtualDir *adds a virtual directory mapping.* 89

4.8.6 int **UpnpRemoveVirtualDir** (IN const char* dirName)
UpnpRemoveVirtualDir *removes a virtual directory mapping made with* **UpnpAddVirtualDir**. 89

4.8.7 void **UpnpRemoveAllVirtualDirs** ( )
UpnpRemoveAllVirtualDirs *removes all virtual directory mappings.* 90

4.8.1

int **UpnpSetWebServerRootDir** (IN const char* rootDir)

UpnpSetWebServerRootDir *sets the document root directory for the internal web server.*

UpnpSetWebServerRootDir sets the document root directory for the internal web server. This directory is considered the root directory (i.e. "/") of the web server.

This function also activates or deactivates the web server. To disable the web server, pass `NULL` for **rootDir**; to activate, pass a valid directory string.

Note that this function is not available when the web server is not compiled into the SDK.

Return Value: `[int]` An integer representing one of the following:

- `UPPN_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_ARGUMENT`: **rootDir** is an invalid directory.

Parameters: `rootDir` Path of the root directory of the web server.

4.8.2

int **UpnpSetVirtualDirCallbacks** (IN struct UpnpVirtualDirCallbacks* callbacks)

UpnpSetVirtualDirCallbacks *sets the callback function to be used to access a virtual directory.*

UpnpSetVirtualDirCallbacks sets the callback function to be used to access a virtual directory. Refer to the description of **UpnpVirtualDirCallbacks** for a description of the functions.

Return Value: `[int]` An integer representing one of the following:

- `UPPN_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_ARGUMENT`: **callbacks** is not a valid pointer.

Parameters: `callbacks` Pointer to a structure containing points to the virtual interface functions.

4.8.3

int **UpnpEnableWebserver** (IN int enable)

UpnpEnableWebServer *enables or disables the webserver.*

UpnpEnableWebServer enables or disables the webserver. A value of `TRUE` enables the webserver, `FALSE` disables it.

Return Value: `[int]` An integer representing one of the following:

- `UPPN_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_ARGUMENT`: **enable** is not valid.

Parameters: `enable` `TRUE` to enable, `FALSE` to disable.

4.8.4

int **UpnpIsWebserverEnabled** ()

UpnpIsWebServerEnabled *returns* `TRUE` *if the webserver is enabled, or* `FALSE` *if it is not.*

UpnpIsWebServerEnabled returns `TRUE` if the webserver is enabled, or `FALSE` if it is not.

Return Value: `[int]` An integer representing one of the following:

- `TRUE`: The webserver is enabled.
- `FALSE`: The webserver is not enabled

4.8.5

int **UpnpAddVirtualDir** (IN const char* dirName)

UpnpAddVirtualDir *adds a virtual directory mapping.*

UpnpAddVirtualDir adds a virtual directory mapping.

All webserver requests containing the given directory are read using functions contained in a **UpnpVirtualDirCallbacks** structure registered via **UpnpSetVirtualDirCallbacks**.

Return Value: `[int]` An integer representing one of the following:

- `UPPN_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_ARGUMENT`: **dirName** is not valid.

Parameters: `dirName` The name of the new directory mapping to add.

4.8.6

int **UpnpRemoveVirtualDir** (IN const char* dirName)

UpnpRemoveVirtualDir *removes a virtual directory mapping made with* **UpnpAddVirtualDir**.

UpnpRemoveVirtualDir removes a virtual directory mapping made with **UpnpAddVirtualDir**.

Return Value: `[int]` An integer representing one of the following:

- `UPPN_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_ARGUMENT`: **dirName** is not valid.

Parameters: `dirName` The name of the virtual directory mapping to remove.

4.8.7

void **UpnpRemoveAllVirtualDirs** ( )

UpnpRemoveAllVirtualDirs *removes all virtual directory mappings.*

UpnpRemoveAllVirtualDirs removes all virtual directory mappings.

Return Value: `[void]` This function does not return a value.

5 Optional Tool APIs

Names

5.1 int **UpnpResolveURL** (IN char* BaseURL, IN char* RelURL, OUT char* AbsURL)
UpnpResolveURL *combines a base URL and a relative URL into a single absolute URL.* 92

5.2 IXML_Document*
UpnpMakeAction (IN char* ActionName, IN char* ServType, IN int NumArg, IN char* Arg, IN ...)
UpnpMakeAction *creates an action request packet based on its input parameters (status variable name and value pair).* . 93

5.3 int **UpnpAddToAction** (IN OUT IXML_Document** ActionDoc, IN char* ActionName, IN char* ServType, IN char* ArgName, IN char* ArgVal)
UpnpAddToAction *creates an action request packet based on its input parameters (status variable name and value pair).* 93

5.4 IXML_Document*
UpnpMakeActionResponse (IN char* ActionName, IN char* ServType, IN int NumArg, IN char* Arg, IN ...)
UpnpMakeActionResponse *creates an action response packet based on its output parameters (status variable name and value pair).* 94

5.5 int **UpnpAddToActionResponse** (IN OUT IXML_Document** ActionResponse, IN char* ActionName, IN char* ServType, IN char* ArgName, IN char* ArgVal)
UpnpAddToActionResponse *creates an action response packet based on its output parameters (status variable name and value pair).* 94

5.6 int **UpnpAddToPropertySet** (IN OUT IXML_Document** PropSet, IN char* ArgName, IN char* ArgVal)

UpnpAddToPropertySet *can be used when an application needs to transfer the status of many variables at once.* 95

5.7 IXML_Document*
UpnpCreatePropertySet (IN int NumArg, IN char* Arg, IN ...)
UpnpCreatePropertySet *creates a property set message packet.* 96

5.8 const char* **UpnpGetErrorMessage** (int errorcode)
UpnpGetErrorMessage *converts an SDK error code into a string error message suitable for display.* 96

The Linux SDK for UPnP Devices contains some additional, optional utility APIs that can be helpful in writing applications using the SDK. These additional APIs can be compiled out in order to save code size in the SDK. Refer to the README for details.

5.1

int **UpnpResolveURL** (IN char* BaseURL, IN char* RelURL, OUT char* AbsURL)

UpnpResolveURL *combines a base URL and a relative URL into a single absolute URL.*

UpnpResolveURL combines a base URL and a relative URL into a single absolute URL. The memory for **AbsURL** needs to be allocated by the caller and must be large enough to hold the **BaseURL** and **RelURL** combined.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: **RelURL** is `NULL`.
- `UPNP_E_INVALID_URL`: The **BaseURL** / **RelURL** combination does not form a valid URL.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`BaseURL`	The base URL to combine.
`RelURL`	The relative URL to **BaseURL**.
`AbsURL`	A pointer to a buffer to store the absolute URL.

5.2

IXML_Document* **UpnpMakeAction** (IN char* ActionName, IN char* ServType, IN int NumArg, IN char* Arg, IN ...)

UpnpMakeAction *creates an action request packet based on its input parameters (status variable name and value pair).*

UpnpMakeAction creates an action request packet based on its input parameters (status variable name and value pair). Any number of input parameters can be passed to this function but every input variable name should have a matching value argument.

Return Value: `[IXML_Document*]` The action node of **Upnp_Document** type or `NULL` if the operation failed.

Parameters:

`ActionName`	The action name.
`ServType`	The service type.
`NumArg`	Number of argument pairs to be passed.
`Arg`	Status variable name and value pair.

5.3

int **UpnpAddToAction** (IN OUT IXML_Document** ActionDoc, IN char* ActionName, IN char* ServType, IN char* ArgName, IN char* ArgVal)

UpnpAddToAction *creates an action request packet based on its input parameters (status variable name and value pair).*

UpnpAddToAction creates an action request packet based on its input parameters (status variable name and value pair). This API is specially suitable inside a loop to add any number input parameters into an existing action. If no action document exists in the beginning then a **Upnp_Document** variable initialized with `NULL` should be passed as a parameter.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: One or more of the parameters are invalid.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`ActionDoc`	A pointer to store the action document node.
`ActionName`	The action name.
`ServType`	The service type.
`ArgName`	The status variable name.
`ArgVal`	The status variable value.

5.4

IXML_Document* **UpnpMakeActionResponse** (IN char* ActionName, IN char* ServType, IN int NumArg, IN char* Arg, IN ...)

UpnpMakeActionResponse *creates an action response packet based on its output parameters (status variable name and value pair).*

UpnpMakeActionResponse creates an action response packet based on its output parameters (status variable name and value pair). Any number of input parameters can be passed to this function but every output variable name should have a matching value argument.

Return Value:

`[IXML_Document*]`	The action node of **Upnp_Document** type or `NULL` if the operation failed.

Parameters:

`ActionName`	The action name.
`ServType`	The service type.
`NumArg`	The number of argument pairs passed.
`Arg`	The status variable name and value pair.

5.5

int **UpnpAddToActionResponse** (IN OUT IXML_Document** ActionResponse, IN char* ActionName, IN char* ServType, IN char* ArgName, IN char* ArgVal)

UpnpAddToActionResponse *creates an action response packet based on its output parameters (status variable name and value pair).*

UpnpAddToActionResponse creates an action response packet based on its output parameters (status variable name and value pair). This API is especially suitable inside a loop to add any number of input parameters into an existing action response. If no action document exists in the beginning, a **Upnp_Document** variable initialized with `NULL` should be passed as a parameter.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: One or more of the parameters are invalid.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`ActionResponse`	Pointer to a document to store the action document node.
`ActionName`	The action name.
`ServType`	The service type.
`ArgName`	The status variable name.
`ArgVal`	The status variable value.

5.6

int **UpnpAddToPropertySet** (IN OUT IXML_Document** PropSet, IN char* ArgName, IN char* ArgVal)

UpnpAddToPropertySet *can be used when an application needs to transfer the status of many variables at once.*

UpnpAddToPropertySet can be used when an application needs to transfer the status of many variables at once. It can be used (inside a loop) to add some extra status variables into an existing property set. If the application does not already have a property set document, the application should create a variable initialized with `NULL` and pass that as the first parameter.

Return Value: `[int]` An integer representing one of the following:

- `UPNP_E_SUCCESS`: The operation completed successfully.
- `UPNP_E_INVALID_PARAM`: One or more of the parameters are invalid.
- `UPNP_E_OUTOF_MEMORY`: Insufficient resources exist to complete this operation.

Parameters:

`PropSet`	A pointer to the document containing the property set document node.
`ArgName`	The status variable name.
`ArgVal`	The status variable value.

5.7

IXML_Document* **UpnpCreatePropertySet** (IN int NumArg, IN char* Arg, IN ...)

UpnpCreatePropertySet *creates a property set message packet.*

UpnpCreatePropertySet creates a property set message packet. Any number of input parameters can be passed to this function but every input variable name should have a matching value input argument.

Return Value: `[IXML_Document*]` `NULL` on failure, or the property-set document node.

Parameters:

`NumArg`	The number of argument pairs passed.
`Arg`	The status variable name and value pair.

5.8

const char* **UpnpGetErrorMessage** (int errorcode)

UpnpGetErrorMessage *converts an SDK error code into a string error message suitable for display.*

UpnpGetErrorMessage converts an SDK error code into a string error message suitable for display. The memory returned from this function should NOT be freed.

Return Value: `[char*]` An ASCII text string representation of the error message associated with the error code.

Parameters:

`errorcode`	The SDK error code to convert.

6 Compile time configuration options

Names

6.1 **THREAD_IDLE_TIME** 97
6.2 **JOBS_PER_THREAD** 97
6.3 **MIN_THREADS** 98
6.4 **MAX_THREADS** 98
6.5 **HTTP_READ_BYTES** 98
6.6 **NUM_SSDP_COPY** 98
6.7 **SSDP_PAUSE** 99
6.8 **WEB_SERVER_BUF_SIZE** 99
6.9 **AUTO_RENEW_TIME** 99
6.10 **CP_MINIMUM_SUBSCRIPTION_TIME** 99
6.11 **MAX_SEARCH_TIME** 99
6.12 **MIN_SEARCH_TIME** 100
6.13 **AUTO_ADVERTISEMENT_TIME** 100
6.14 **SSDP_PACKET_DISTRIBUTE** 100
6.15 **Module Exclusion** 100
6.16 **DEBUG_LEVEL** 101
6.17 **DEBUG_TARGET** 101

The Linux SDK for UPnP Devices contains some compile-time parameters that effect the behavior of the SDK. All configuration options are located in `inc/config.h`.

6.1 THREAD_IDLE_TIME

The `THREAD_IDLE_TIME` constant determines when a thread will be removed from the thread pool and returned to the operating system. When a thread in the thread pool has been idle for this number of milliseconds the thread will be released from the thread pool. The default value is 5000 milliseconds (5 seconds).

6.2 JOBS_PER_THREAD

The `JOBS_PER_THREAD` constant determines when a new thread will be allocated to the thread pool inside the SDK. The thread pool will try and maintain this jobs/thread ratio. When the

jobs/thread ratio becomes greater than this, then a new thread (up to the max) will be allocated to the thread pool. The default ratio is 10 jobs/thread.

6.3 MIN_THREADS

The `MIN_THREADS` constant defines the minimum number of threads the thread pool inside the SDK will create. The thread pool will always have this number of threads. These threads are used for both callbacks into applications built on top of the SDK and also for making connections to other control points and devices. This number includes persistent threads. The default value is two threads.

6.4 MAX_THREADS

The `MAX_THREADS` constant defines the maximum number of threads the thread pool inside the SDK will create. These threads are used for both callbacks into applications built on top of the library and also for making connections to other control points and devices. It is not recommended that this value be below 10, since the threads are necessary for correct operation. This value can be increased for greater performance in operation at the expense of greater memory overhead. The default value is 12.

6.5 HTTP_READ_BYTES

HTTP responses will read at most `HTTP_READ_BYTES`. This prevents devices that have a misbehaving web server to send a large amount of data to the control point causing it to crash. A value of -1 means there is no max. The default is -1.

6.6 NUM_SSDP_COPY

This configuration parameter determines how many copies of each SSDP advertisement and search packets will be sent. By default it will send two copies of every packet.

6.7 SSDP_PAUSE

This configuration parameter determines the pause between identical SSDP advertisement and search packets. The pause is measured in milliseconds and defaults to 100.

6.8 WEB_SERVER_BUF_SIZE

This configuration parameter sets the maximum buffer size for the webserver. The default value is 1MB.

6.9 AUTO_RENEW_TIME

The `AUTO_RENEW_TIME` is the time, in seconds, before a subscription expires that the SDK automatically resubscribes. The default value is 10 seconds. Setting this value too low can result in the subscription renewal not making it to the device in time, causing the subscription to timeout. In order to avoid continually resubscribing the minimum subscription time is five seconds more than the auto renew time.

6.10 CP_MINIMUM_SUBSCRIPTION_TIME

The `CP_MINIMUM_SUBSCRIPTION_TIME` is the minimum subscription time allowed for a control point using the SDK. Subscribing for less than this time automatically results in a subscription for this amount. The default value is 5 seconds more than the `AUTO_RENEW_TIME`, or 15 seconds.

6.11 MAX_SEARCH_TIME

The `MAX_SEARCH_TIME` is the maximum time allowed for an SSDP search by a control point. Searching for greater than this time automatically results in a search for this amount. The default value is 80 seconds.

6.12 MIN_SEARCH_TIME

The MIN_SEARCH_TIME is the minimumm time allowed for an SSDP search by a control point. Searching for less than this time automatically results in a search for this amount. The default value is 2 seconds.

6.13 AUTO_ADVERTISEMENT_TIME

The AUTO_ADVERTISEMENT_TIME is the time, in seconds, before an device advertisements expires before a renewed advertisement is sent. The default time is 30 seconds.

6.14 SSDP_PACKET_DISTRIBUTE

The SSDP_PACKET_DISTRIBUTE enables the SSDP packets to be sent at an interval equal to half of the expiration time of SSDP packets minus the AUTO_ADVERTISEMENT_TIME. This is used to increase the probability of SSDP packets reaching to control points. It is recommended that this flag be turned on for embedded wireless devices.

6.15 Module Exclusion

Depending on the requirements, the user can selectively discard any of the major modules like SOAP, GENA, SSDP or the Internal web server. By default everything is included inside the SDK. By setting any of the values below to 0, that component will not be included in the final SDK.

- EXCLUDE_SOAP[0,1]
- EXCLUDE_GENA[0,1]
- EXCLUDE_SSDP[0,1]
- EXCLUDE_DOM [0,1]
- EXCLUDE_WEB_SERVER[0,1]
- EXCLUDE_JNI[0,1]

6.16 DEBUG_LEVEL

The user has the option to select 4 different types of debugging levels. The critical level (3) will show only those messages which can halt the normal processing of the library, like memory allocation errors. The remaining three levels are just for debugging purposes. Packet level will display all the incoming and outgoing packets that are flowing between the control point and the device. Info Level displays the other important operational information regarding the working of the library. If the user selects All, then the library displays all the debugging information that it has.

- `Critical [0]`
- `Packet Level [1]`
- `Info Level [2]`
- `All [3]`

6.17 DEBUG_TARGET

The user has the option to redirect the library output debug messages to either the screen or to a log file. All the output messages with debug level 0 will go to `upnp.err` and messages with debug level greater than zero will be redirected to `upnp.out`.

7 Other debugging features

Names

7.1 **DBGONLY** 102

The UPnP SDK contains other features to aid in debugging.

7.1 DBGONLY

The **DBGONLY** macro allows code to be marked so that it is only included in the DEBUG build and not the release. To use this macro, put the code inside of the parentheses:

```
DBGONLY(int i;)
```

This will cause a declaration of the integer `i` only in the debug build.