EBS UPNP Control Point SDK

Contents

 Embedded UPnP Library version 1.0

1 API Reference

Names

1.1 int **UPnP_RuntimeInit** (UPnPRuntime* rt, UPNP_UINT8* serverAddr, UPNP_UINT16 serverPort, UPNP_INT16 ipType, UPNP_CHAR* wwwRootDir, UPNP_INT16 maxConnections, UPNP_INT16 maxHelperThreads)
Initialize a UPnPRuntime ..... **??**

1.2 void **UPnP_RuntimeDestroy** (UPnPRuntime* rt)
Destroy a UPnPRuntime ..... **??**

1.3 int **UPnP_AddVirtualFile** (UPnPRuntime* rt, const UPNP_CHAR* serverPath, const UPNP_UINT8* data, UPNP_INT32 size, const UPNP_CHAR* contentType)
Create a virtual file on the HTTP server. ..... **??**

1.4 int **UPnP_RemoveVirtualFile** (UPnPRuntime* rt, const UPNP_CHAR* serverPath)
Remove a virtual file from the server ..... **??**

1.5 int **UPnP_ProcessState** (UPnPRuntime* rt, UPNP_INT32 msecTimeout)
Process asynchronous operations in non-threaded mode. ..... **??**

1.6 int **UPnP_StartDaemon** (UPnPRuntime* rt)
Start the UPnP Daemon thread.

??

1.7 int **UPnP_StopDaemon** (UPnPRuntime* rt, UPNP_INT32 secTimeout)

Kill the UPnP Daemon thread. **??**

1.8 const UPNP_CHAR*
UPnP_GetPropertyValueByName (IXML_Document* propertySet, const UPNP_CHAR* name)
Get the value of a named property in a GENA notify message. ... **??**

1.9 const UPNP_CHAR*
UPnP_GetPropertyNameByIndex (IXML_Document* propertySet, int index)
Get the name of the nth property. ..... **??**

1.10 const UPNP_CHAR*
UPnP_GetPropertyValueByIndex (IXML_Document* propertySet, int index)
Get the value of the nth property. ..... **??**

1.11 int **UPnP_AddToPropertySet** (IXML_Document** doc, const UPNP_CHAR* name, const UPNP_CHAR* value)
Add name and value pair to GENA notify message property set. ..... **??**

1.12 int **UPnP_CreateActionResponse** (UPnPActionRequest* request)
Creates a SOAP action response message. ..... **??**

1.13 IXML_Document*

UPnP_CreateAction (const UPNP_CHAR* serviceTypeURI, const UPNP_CHAR* actionName)
Create a SOAP action request. **??**

1.14 int **UPnP_SetActionArg** (IXML_Document* actionDoc, const UPNP_CHAR* name, const UPNP_CHAR* value)
Set an argument for a SOAP action response/request ..... **??**

1.15 int **UPnP_ControlPointInit** (UPnPControlPoint* cp, UPnPRuntime* rt, UPnPControlPointCallback callbackFn, void* callbackData)
Initialize a UPnPControlPoint **??**

1.16 void **UPnP_ControlPointDestroy** (UPnPControlPoint* cp, UPNP_INT32 gracefulTimeoutMsec)
Destroy a UPnPControlPoint .. **??**

1.17 int **UPnP_ControlFindAll** (UPnPControlPoint* cp, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)
Search for UPnP devices. ..... **??**

1.18 int **UPnP_ControlFindAllDevices** (UPnPControlPoint* cp, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)

Search for a particular UPnP device. ..... **??**

1.21 int **UPnP_ControlFindDevicesByService** (UPnPControlPoint* cp, UPNP_CHAR* serviceType, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)

Search for UPnP devices that offer a certain service. ..... **??**

1.22 UPnPControlDeviceHandle **UPnP_ControlOpenDevice** (UPnPControlPoint* cp, UPNP_CHAR* url)

Open a remote device for description, control, and eventing.

??

1.23 int **UPnP_ControlOpenDeviceAsync** (UPnPControlPoint* cp, UPNP_CHAR* url, void* userData)

Asynchronously open a remote device for description, control, and eventing. ..... **??**

1.24 void **UPnP_ControlCloseDevice** (UPnPControlDeviceHandle deviceHandle)

Close an open device handle ... **??**

1.25 int **UPnP_ControlGetDeviceInfo** (UPnPControlDeviceHandle device, UPnPControlDeviceInfo* info)

Get information for an open device. ..... **??**

1.26 int **UPnP_ControlGetServiceOwnerDeviceInfo** (UPnPControlDeviceHandle handle, UPNP_CHAR* serviceId, UPnPControlDeviceInfo* info)

Get information for the parent device of a service ..... **??**

1.27 UPNP_CHAR*

UPnP_ControlGetServiceType (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId)

Get the type of a service ..... **??**

1.28 int **UPnP_ControlGetServices** (UPnPControlDeviceHandle deviceHandle, UPnPControlServiceIterator* i)

Enumerate the services on a device ..... **??**

1.29 int **UPnP_ControlGetServicesByType** (UPnPControlDeviceHandle deviceHandle, UPnPControlServiceIterator* i, UPNP_CHAR* serviceType)

Enumerate the services of a certain type on a device ..... **??**

1.30 UPNP_CHAR*

UPnP_ControlNextService (UPnPControlServiceIterator* i)

Enumerate the next service on the device. ..... **??**

1.31 void **UPnP_ControlServiceIteratorDone** (UPnPControlServiceIterator* i)

Clean up when done enumerating services. ..... **??**

1.32 IXML_Document*

UPnP_ControlGetServiceInfo (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId)

Get detailed information about a service. ..... **??**

1.33 int **UPnP_ControlGetServiceInfoAsync** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, void* userData)

Asynchronous get detailed information about a service. ..... **??**

1.34 int **UPnP_ControlInvokeAction** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, const UPNP_CHAR* actionName, IXML_Document* action, void* userData, UPNP_BOOL waitForCompletion)

Invoke an action on a remote service ..... **??**

1.35 int **UPnP_ControlSubscribe** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, UPNP_INT32 timeoutSec, void* userData, UPNP_BOOL waitForCompletion)

Subscribe to a service or renew a subscription ..... **??**

1.36 UPNP_BOOL **UPnP_ControlSubscribedToService** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId)

Return whether or not the control point is subscribed to the given service ..... **??**

1.37 void **UPnP_ControlSetServiceSubscriptionExpireWarning** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, UPNP_INT32 warningMsec)

Sets the time offset before subscription expiration at which a warning event will be generated.

??

1.38 int **UPnP_ControlUnsubscribe** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, void* userData, UPNP_BOOL waitForCompletion)

Cancel a subscribed service .... **??**

1.1

int **UPnP_RuntimeInit** (UPnPRuntime* rt, UPNP_UINT8* serverAddr, UPNP_UINT16 serverPort, UPNP_INT16 ipType, UPNP_CHAR* wwwRootDir, UPNP_INT16 maxConnections, UPNP_INT16 maxHelperThreads)

Initialize a UPnPRuntime

Initializes the given UPnPRuntime struct, and sets up an HTTP server instance to receive control/event messages. This function must be called before any other function in the UPnP SDK.

Return Value: `error` code

Parameters:	`rt`	pointer to uninitialized UPnPRuntime struct
	`serverAddr`	ip address to bind HTTP server to (NULL for IP_ADDR_ANY)
	`serverPort`	port to bind HTTP server to (0 for ANY_PORT)
	`ipType`	type of ip version used (ipv4 or ipv6), (RTP_NET_TYPE_IPV4 for v4 and RTP_NET_TYPE_IPV6 for v6)
	`wwwRootDir`	HTTP root dir on local file system
	`maxConnections`	the maximum limit on simultaneous HTTP server connections
	`maxHelperThreads`	if UPNP_MULTITHREAD is defined, the max number of helper threads to spawn

1.2

void **UPnP_RuntimeDestroy** (UPnPRuntime* rt)

Destroy a UPnPRuntime

Must be called after all other UPnP SDK calls to clean up runtime data for UPnP.

Return Value:	`error` code
Parameters:	`rt` pointer to UPnPRuntime struct

1.3

int **UPnP_AddVirtualFile** (UPnPRuntime* rt, const UPNP_CHAR* serverPath, const UPNP_UINT8* data, UPNP_INT32 size, const UPNP_CHAR* contentType)

Create a virtual file on the HTTP server.

Makes the data buffer passed in available at the given path on the HTTP server.

Return Value: `error` code
See Also: UPnP_RemoveVirtualFile

1.4

int **UPnP_RemoveVirtualFile** (UPnPRuntime* rt, const UPNP_CHAR* serverPath)

Remove a virtual file from the server

Must be called before UPnP_RuntimeDestroy to remove any virtual files added using UPnP_AddVirtualFile.

Return Value: `error` code

1.5

int **UPnP_ProcessState** (UPnPRuntime* rt, UPNP_INT32 msecTimeout)

Process asynchronous operations in non-threaded mode.

This function blocks for at most msecTimeout milliseconds, processing any asynchronous operations that may be in progress on either the control point or device runtime attached to the given UPnPRuntime.

This function must be called in order to receive events if an application is running with the UPnP SDK in single-threaded mode.

Return Value:	`error`	code
Parameters:	`rt`	pointer to UPnPRuntime struct
	`msecTimeout`	time in miliseconds for which the function blocks

1.6

int **UPnP_StartDaemon** (UPnPRuntime* rt)

Start the UPnP Daemon thread.

This function must be called in multithreaded mode to start the UPnP daemon, which listens for requests/announcements on the network, and sends any events to the attached control point/device runtime.

Return Value:	`error` code
Parameters:	`rt` pointer to UPnPRuntime struct
See Also:	UPnP_StopDaemon

1.7

int **UPnP_StopDaemon** (UPnPRuntime* rt,
UPNP_INT32 secTimeout)

Kill the UPnP Daemon thread.

This function stops the UPnP daemon from executing. It will wait for at most secTimeout seconds for all helper threads to terminate. If this function returns negative error code, it means the timeout expired without the successful termination of one or more helper threads. In this case, calling UPnP_RuntimeDestroy may cause a fault since there are still helper threads

running that may try to access the data structures pointed to by the UPnPRuntime.

Return Value:	`error`	code
Parameters:	`rt`	the device runtime to stop
	`secTimeout`	time to wait for daemon to stop

1.8

const UPNP_CHAR* **UPnP_GetPropertyValueByName** (IXML_Document* propertySet, const UPNP_CHAR* name)

Get the value of a named property in a GENA notify message.

The string returned must not be modified in any way. It is valid until the IXML_Document is deleted.

Return Value:	`the`	value or NULL if the property was not found

1.9

const UPNP_CHAR* **UPnP_GetPropertyNameByIndex** (IXML_Document* propertySet, int index)

Get the name of the nth property.

The string returned must not be modified in any way. It is valid until the IXML_Document is deleted.

Return Value:	`the`	value or NULL if the property was not found
Parameters:	`propertySet`	address of xml property set
	`index`	index in property for value

1.10

const UPNP_CHAR* **UPnP_GetPropertyValueByIndex** (IXML_Document* propertySet, int index)

Get the value of the nth property.

The string returned must not be modified in any way. It is valid until the IXML_Document is deleted.

Return Value:	`the`	value or NULL if the property was not found
Parameters:	`propertySet`	address of xml property set
	`index`	index in property for value

1.11

int **UPnP_AddToPropertySet** (IXML_Document** doc, const UPNP_CHAR* name, const UPNP_CHAR* value)

Add name and value pair to GENA notify message property set.

Add a new name value pair entry to the property set

Return Value:	`error`	code
Parameters:	`doc`	address of property set
	`name`	pointer to name for new entry
	`value`	address of value of for the new entry

1.12

int **UPnP_CreateActionResponse** (UPnPActionRequest* request)

Creates a SOAP action response message.

Creates a response message skeleton for the supplied SOAP action request

Return Value: `error` code

1.13

IXML_Document* **UPnP_CreateAction** (const UPNP_CHAR* serviceTypeURI, const UPNP_CHAR* actionName)

Create a SOAP action request.

Creates an XML document which will hold the SOAP action request message. This function returns the address of newly formed XML document. After finishing the process of sending action request the application must release this xml document.

Return Value:	`pointer`	to newly created IXML_Document, which can be passed intoUPnP_SetActionArg to set the action arguments; NULL on error
Parameters:	`serviceTypeURI`	string containing service type of the target service
	`actionName`	name on action on the target service

1.14

int **UPnP_SetActionArg** (IXML_Document* actionDoc,
const UPNP_CHAR* name,
const UPNP_CHAR* value)

Set an argument for a SOAP action response/request

This function can be used on an IXML_Document created by either UPnP_CreateActionResponse (→ 1.13, *page* **??**) or UPnP_CreateAction to set either the input or output arguments for a SOAP action.

Return Value:	`error`	code
Parameters:	`actionDoc`	pointer to action respose message
	`name`	argument name *
	`value`	argument value *

1.15

int **UPnP_ControlPointInit** (UPnPControlPoint* cp,
UPnPRuntime* rt, UPnPControlPointCallback callbackFn, void* callbackData
)

Initialize a UPnPControlPoint

Initializes all control point state data in a UPnPControlPoint struct (allocated by the calling application), and binds the control point to the specified UPnPRuntime. The UPnPRuntime must be initialized via UPnP_RuntimeInit before this function is called. Only one control point may be bound to a single UPnPRuntime at once. This function must be called before all other control point related functions.

Return Value:	`error` code	
Parameters:	`cp`	pointer to control pointcontext instance
	`rt`	UPnP runtime to associate withthis control point
	`callbackFn`	callback for this controlpoint
	`callbackData`	application-specific datawhich will be passed into thecallback
See Also:	UPnP_ControlPointDestroy	

1.16

void **UPnP_ControlPointDestroy** (UPnPControlPoint* cp, UPNP_INT32 gracefulTimeoutMsec)

Destroy a UPnPControlPoint

Cleans up all data associated with a UPnPControlPoint structure. Once this function has been called, it is safe to free the memory used by the UPnPControlPoint structure.

Return Value:	`error` code	
Parameters:	`cp`	the control point to destroy
	`gracefulTimeoutMsec`	if this control point has anyoutstanding operations, wait forthis many milliseconds to allow themto complete gracefully. Aftertimeout expires, do hard close.
See Also:	UPnP_ControlPointInit	

1.17

int **UPnP_ControlFindAll** (UPnPControlPoint* cp, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)

Search for UPnP devices.

Sends a request for all UPnP devices on the network to make their presence known to the control point. If this search method is used, then seperate UPNP_CONTROL_EVENT_DEVICE_FOUND events will be generated for each root device, embedded device, and service that responds.

When the timeout has been reached, a UPNP_CONTROL_EVENT_SEARCH_COMPLETE

event will be sent to the control point.

Generates: UPNP_CONTROL_EVENT_DEVICE_FOUND UPNP_CONTROL_EVENT_SEARCH_COMPLETE

Return Value: `error` code

Parameters:

`cp`	control point, initialized byUPnP_ControlPointInit
`timeoutMsec`	total duration of time, in milliseconds, for the search
`userData`	passed to callback as userRequestDatafor UPNP_CONTROL_EVENT_DEVICE_FOUNDevent
`waitForCompletion`	if UPNP_TRUE, this function will notreturn until the search completes; elsethe function will return immediatelyafter sending the multicast searchrequest.

See Also: UPnP_ControlFindAllDevices

1.18

int **UPnP_ControlFindAllDevices** (UPnPControlPoint* cp, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)

Search for UPnP devices.

Sends a request for all UPnP devices on the network to make their presence known to the control point. Only one UPNP_CONTROL_EVENT_DEVICE_FOUND event per root device will be generated if this function is used.

When the timeout has been reached, a UPNP_CONTROL_EVENT_SEARCH_COMPLETE

event will be sent to the control point.

Generates: UPNP_CONTROL_EVENT_DEVICE_FOUND UPNP_CONTROL_EVENT_SEARCH_COMPLETE

Return Value:	`error` code	
Parameters:	`cp`	control point, initialized byUPnP_ControlPointInit
	`timeoutMsec`	total duration of time, in milliseconds, for the search
	`userData`	passed to callback as userRequestDatafor UPNP_CONTROL_EVENT_DEVICE_FOUNDevent
	`waitForCompletion`	if UPNP_TRUE, this function will notreturn until the search completes; elsethe function will return immediatelyafter sending the multicast searchrequest.
See Also:	UPnP_ControlFindAll	

1.19

int **UPnP_ControlFindDevicesByType** (UPnPControlPoint* cp, UPNP_CHAR* deviceType, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)

Search for UPnP devices of a certain type.

Sends a request for all UPnP devices of a certain type on the network to make their presence known to the control point. One

UPNP_CONTROL_EVENT_DEVICE_FOUND event per device that matches the search type will be generated if this function is used.

When the timeout has been reached, a UPNP_CONTROL_EVENT_SEARCH_COMPLETE

event will be sent to the control point.

Generates: UPNP_CONTROL_EVENT_DEVICE_FOUND UPNP_CONTROL_EVENT_SEARCH_COMPLETE

Return Value: `error` code

Parameters:

`cp`	control point, initialized byUPnP_ControlPointInit
`deviceType`	device type to search for, as specifiedby the UPnP Forum.
`timeoutMsec`	total duration of time, in milliseconds, for the search
`userData`	passed to callback as userRequestDatafor UPNP_CONTROL_EVENT_DEVICE_FOUNDevent
`waitForCompletion`	if UPNP_TRUE, this function will notreturn until the search completes; elsethe function will return immediatelyafter sending the multicast searchrequest.

See Also: UPnP_ControlFindDevicesByUUID, UPnP_ControlFindDevicesByService

1.20

int **UPnP_ControlFindDevicesByUUID** (UPnPControlPoint* cp, UPNP_CHAR* uuid, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)

Search for a particular UPnP device.

Sends a request for UPnP devices with the given UUID(unique identifier) to make their presence known to the control point. One UPNP_CONTROL_EVENT_DEVICE_FOUND event per device that matches the search type will be generated if this function is used.

When the timeout has been reached, a UPNP_CONTROL_EVENT_SEARCH_COMPLETE

event will be sent to the control point.

Generates: UPNP_CONTROL_EVENT_DEVICE_FOUND UPNP_CONTROL_EVENT_SEARCH_COMPLETE

Return Value: `error` code

Parameters:

`cp`	control point, initialized byUPnP_ControlPointInit
`uuid`	uuid to search for, as specified by the-UPnP Forum.
`timeoutMsec`	total duration of time, in milliseconds, for the search
`userData`	passed to callback as userRequestDatafor UPNP_CONTROL_EVENT_DEVICE_FOUNDevent
`waitForCompletion`	if UPNP_TRUE, this function will notreturn until the search completes; elsethe function will return immediatelyafter sending the multicast searchrequest.

See Also: UPnP_ControlFindDevicesByType

1.21

int **UPnP_ControlFindDevicesByService** (UPnPControlPoint* cp, UPNP_CHAR* serviceType, UPNP_INT32 timeoutMsec, void* userData, UPNP_BOOL waitForCompletion)

Search for UPnP devices that offer a certain service.

Sends a request for UPnP devices with one or more services of the given type to make their presence known to the control point. One UPNP_CONTROL_EVENT_DEVICE_FOUND event per device that matches the search type will be generated if this function is used.

When the timeout has been reached, a UPNP_CONTROL_EVENT_SEARCH_COMPLETE

event will be sent to the control point.

Generates: UPNP_CONTROL_EVENT_DEVICE_FOUND UPNP_CONTROL_EVENT_SEARCH_COMPLETE

Return Value:	`error` code	
Parameters:	`cp`	control point, initialized byUPnP_ControlPointInit
	`serviceType`	service type to search for, asspecified by the UPnP Forum.
	`timeoutMsec`	total duration of time, in milliseconds,for the search
	`userData`	passed to callback as userRequestDatafor UPNP_CONTROL_EVENT_DEVICE_FOUNDevent
	`waitForCompletion`	if UPNP_TRUE, this function will notreturn until the search completes; elsethe function will return immediatelyafter sending the multicast searchrequest.
See Also:	UPnP_ControlFindDevicesByType	

1.22

UPnPControlDeviceHandle **UPnP_ControlOpenDevice** (UPnPControlPoint* cp, UPNP_CHAR* url)

Open a remote device for description, control, and eventing.

Opens the device at the specified URL. This function is used to obtain a UPnPControlDeviceHandle, which is used for most functions which operate within the UPnP description, control, and eventing phases.

This function will block until the device open has completed or failed.

Generates: (no events)

Return Value:	`UPnPControlDeviceHandle`	for the open device, or 0 on failure
Parameters:	`cp`	control point, initialized byUPnP_ControlPointInit
	`url`	url of device to open

See Also: UPnP_ControlOpenDeviceAsync

1.23

int **UPnP_ControlOpenDeviceAsync** (UPnPControlPoint* cp, UPNP_CHAR* url, void* userData)

Asynchronously open a remote device for description, control, and eventing.

Opens the device at the specified URL. This function is used to obtain a UPnPControlDeviceHandle, which is used for most functions which operate within the UPnP description, control, and eventing phases.

This function will return immediately if the control point is able to initiate the device open successfully. One of the events listed below will be sent to the control point once the open completes (and the UPnPControlDeviceHandle is available) or an error occurs.

Generates: UPNP_CONTROL_EVENT_DEVICE_OPEN UPNP_CONTROL_EVENT_DEVICE_OPEN_FAILED

Return Value: `UPnPControlDeviceHandle` for the open device, or 0 on failure

Parameters:

`cp`	control point, initialized byUPnP_ControlPointInit
`url`	url of device to open
`userData`	passed to callback as userRequestData forUPNP_CONTROL_EVENT_DEVICE_OPEN event

See Also: UPnP_ControlOpenDeviceAsync

1.24

void **UPnP_ControlCloseDevice** (UPnPControlDeviceHandle deviceHandle)

Close an open device handle

Closes a device opened with UPnP_ControlOpenDevice or UPnP_ControlOpenDeviceAsync (→ 1.22, *page* **??**). The device handle passed in may not be used after this function is called.

Generates: (no events)

Return Value: `UPnPControlDeviceHandle` for the open device, or 0 on failure
Parameters: `deviceHandle` handle returned byUPnP_ControlOpenDevice
See Also: UPnP_ControlOpenDevice, UPnP_ControlOpenDeviceAsync

1.25

int **UPnP_ControlGetDeviceInfo** (UPnPControlDeviceHandle device, UPnPControlDeviceInfo* info)

Get information for an open device.

Populates the fields of the specified UPnPControlDeviceInfo structure with various information about the given device, such as device type, UDN, manufacturer, model number, etc.

Generates: (no events)

Return Value: `error` code
Parameters: `device` handle returned byUPnP_ControlOpenDevice
`info` UPnPControlDeviceInfo to populate
See Also: UPnP_ControlGetServiceOwnerDeviceInfo

1.26

int **UPnP_ControlGetServiceOwnerDeviceInfo** (UPnPControlDeviceHandle handle, UPNP_CHAR* serviceId, UPnPControlDeviceInfo* info)

Get information for the parent device of a service

Populates the fields of the specified UPnPControlDeviceInfo structure with various information about the parent device of the given service. By contrast, UPnP_ControlGetDeviceInfo gets information about the root device ONLY. This function is useful for gathering information about embedded UPnP devices.

Generates: (no events)

Return Value:	`error`	code
Parameters:	`handle`	handle returned byUPnP_ControlOpenDevice
	`serviceId`	serviceId of service for whoseparent to get info
	`info`	UPnPControlDeviceInfo topopulate
See Also:	UPnP_ControlGetDeviceInfo, UPnP_ControlGetServiceType	

1.27

UPNP_CHAR* **UPnP_ControlGetServiceType** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId)

Get the type of a service

Returns the service type (as defined by the UPnP Forum) of the given service instance on an open device. The string passed back by this function must not be modified in any way. It is valid until UPnP_ControlDeviceClose is called on the given device handle.

Generates: (no events)

Return Value:	`error`	code
Parameters:	`deviceHandle`	handle returned by UPnP_ControlOpenDevice
	`serviceId`	serviceId of service to gettype for
See Also:	UPnP_ControlDeviceClose, UPnP_ControlGetServiceOwnerDeviceInfo	

1.28

int **UPnP_ControlGetServices** (UPnPControlDeviceHandle deviceHandle, UPnPControlServiceIterator* i)

Enumerate the services on a device

Initializes a UPnPControlServiceIterator to enumerate all the services on all embedded devices on the given device.

Generates: (no events)

Return Value:	`error`	code
Parameters:	`deviceHandle`	handle returned by UPnP_ControlOpenDevice
	`i`	uninitialized UPnPControlServiceIteratorto use for this enumeration
See Also:	UPnP_ControlGetServicesByType, UPnP_ControlNextService, UPnP_ControlServiceIteratorDone	

1.29

int **UPnP_ControlGetServicesByType** (UPnPControlDeviceHandle deviceHandle, UPnPControlServiceIterator* i, UPNP_CHAR* serviceType)

Enumerate the services of a certain type on a device

Initializes a UPnPControlServiceIterator to enumerate all the services of the given type (as defined by the UPnP Forum) on the given device.

Generates: (no events)

Return Value:	`error` code	
Parameters:	`deviceHandle`	handle returned by UPnP_ControlOpenDevice
	`i`	uninitialized UPnPControlServiceIteratorto use for this enumeration
	`serviceType`	type of service toenumerate
See Also:	UPnP_ControlGetServices, UPnP_ControlNextService, UPnP_ControlServiceIteratorDone	

1.30

UPNP_CHAR* **UPnP_ControlNextService** (UPnPControlServiceIterator* i)

Enumerate the next service on the device.

Returns the unique serviceId of the next service instance in the given enumeration (initialized by UPnP_ControlGetServices or UPnP_ControlGetServicesByType (→ 1.28, *page* **??**)). The string returned by this function must not be modified in any way, and is valid until UPnP_ControlDeviceClose is called for this device.

Generates: (no events)

Return Value:	`serviceId`	if there is a next service; otherwise NULL
Parameters:	`i`	UPnPControlServiceIteratorinitialized by UPnP_ControlGetServices or UPnP_ControlGetServicesByType (→ 1.28, *page* **??**)

See Also: UPnP_ControlGetServices,
UPnP_ControlGetServicesByType,
UPnP_ControlServiceIteratorDone

1.31

void **UPnP_ControlServiceIteratorDone** (UPnPControlServiceIterator* i)

Clean up when done enumerating services.

Must be called when done enumerating services using UPnP_ControlNextService.

Generates: (no events)

Return Value: `nothing`
Parameters: `i` UPnPControlServiceIteratorinitialized by UPnP_ControlGetServices or UPnP_ControlGetServicesByType (→ 1.28, *page* **??**)
See Also: UPnP_ControlGetServices,
UPnP_ControlGetServicesByType,
UPnP_ControlNextService

1.32

IXML_Document* **UPnP_ControlGetServiceInfo** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId)

Get detailed information about a service.

This function retrieves the service description document(SCPD) from the given device and parses it into an XML DOM structure. The resulting

IXML_Document returned by this function is owned by the caller of this function. See the UPnP Forum SCPD definition for more information on the content and structure of this document.

When this function returns, the document has either successfully loaded, or an error has occurred

Generates: (no events)

Return Value:	`XML`	DOM tree for the given document, or NULL on error
Parameters:	`deviceHandle`	handle returned by UPnP_ControlOpenDevice
	`serviceId`	id of service to get detailedinfo for
See Also:	UPnP_ControlGetServiceInfoAsync	

1.33

int **UPnP_ControlGetServiceInfoAsync** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, void* userData)

Asynchronous get detailed information about a service.

This function retrieves the service description document (SCPD) from the given device and parses it into an XML DOM structure. The IXML_Document is passed back to the control point through a UPNP_CONTROL_EVENT_SERVICE_INFO_READ event.

This function will return immediately; an event from the list below is sent to the control point when the SCPD has been downloaded, or the operation fails due to some error.

Generates: UPNP_CONTROL_EVENT_SERVICE_INFO_READ, UPNP_CONTROL_EVENT_SERVICE_GET_INFO_FAILED

Return Value:	`error`	code
Parameters:	`deviceHandle`	handle returned by UPnP_ControlOpenDevice
	`serviceId`	id of service to get detailedinfo for
	`userData`	passed to callback asuserRequestData for generatedevents

See Also: UPnP_ControlGetServiceInfo

1.34

int **UPnP_ControlInvokeAction** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, const UPNP_CHAR* actionName, IXML_Document* action, void* userData, UPNP_BOOL waitForCompletion)

Invoke an action on a remote service

This function will send the given action to a service on a remote device and generate an event that contains the response to that action. If waitForCompletion is UPNP_TRUE, then UPnP_ControlInvokeAction will not return until the UPNP_CONTROL_EVENT_ACTION_COMPLETE event is passed to the control point callback. Otherwise, it will return immediately and the event is sent to the control point when the action completes or an error occurs.

The action passed into this function must be generated using UPnP_CreateAction and UPnP_SetActionArg.

Generates: UPNP_CONTROL_EVENT_ACTION_COMPLETE

Return Value: `error` code

Parameters:

`deviceHandle`	handle returned by UPnP_ControlOpenDevice
`serviceId`	id of service to invokeaction on
`actionName`	name of action to invoke
`action`	SOAP action message createdusing UPnP_CreateActionand UPnP_SetActionArg
`userData`	passed to callback asuserRequestData for generatedevents
`waitForCompletion`	see above description

See Also:

1.35

int **UPnP_ControlSubscribe** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, UPNP_INT32 timeoutSec, void* userData, UPNP_BOOL waitForCompletion)

Subscribe to a service or renew a subscription

Subscribes to a service on a remote device to receive notifications when the service's state changes. If the control point is already subscribed to the given service, this function has the effect of renewing the existing subscription for the given period of time.

If waitForCompletion is UPNP_TRUE, this function will wait until the subscription request has been either accepted or rejected. Otherwise it returns immediately and a UPNP_CONTROL_EVENT_SUBSCRIPTION_ACCEPTED or UPNP_CONTROL_EVENT_SUBSCRIPTION_REJECTED event is generated once the device responds.

Once the control point is subscribed to service, UPNP_CONTROL_EVENT_SERVICE_STATE_UPDATE events may be generated to indicate the service's state has been updated. If such a notification is dropped, a UPNP_CONTROL_EVENT_SUBSCRIPTION_OUT_OF_SYNC event will be sent to the control point, which gives the application the opportunity to indicate through the callback return value whether to drop the subscription or attempt to re-subscribe. If the application chooses to re-subscribe (synchronize), a UPNP_CONTROL_EVENT_SYNCHRONIZE_FAILED event may be generated, if the synchronization fails. If the UPNP_CONTROL_EVENT_SYNCHRONIZE_FAILED is not handled by the control point callback, the default action is to try to synchronize the subscription by re-subscribing.

When the subscription is close to expiring, the control point will receive a UPNP_CONTROL_EVENT_SUBSCRIPTION_NEAR_EXPIRATION event, to give the application a chance to renew the subscription.

Once subscription expires, a UPNP_CONTROL_EVENT_SUBSCRIPTION_EXPIRED event is sent to the control point.

Generates: UPNP_CONTROL_EVENT_SUBSCRIPTION_ACCEPTED, UPNP_CONTROL_EVENT_SUBSCRIPTION_REJECTED, UPNP_CONTROL_EVENT_SERVICE_STATE_UPDATE, UPNP_CONTROL_EVENT_SUBSCRIPTION_OUT_OF_SYNC, UPNP_CONTROL_EVENT_SYNCHRONIZE_FAILED, UPNP_CONTROL_EVENT_SUBSCRIPTION_NEAR_EXPIRATION, UPNP_CONTROL_EVENT_SUBSCRIPTION_EXPIRED

Return Value: `error` code

Parameters:

`deviceHandle`	handle returned by UPnP_ControlOpenDevice
`serviceId`	id of service to subscribeto
`timeoutSec`	duration in seconds for thesubscription
`userData`	passed to callback asuserRequestData for generatedevents
`waitForCompletion`	see above description

See Also: UPnP_ControlUnsubscribe, UPnP_ControlSubscribedToService

1.36

UPNP_BOOL **UPnP_ControlSubscribedToService** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId)

Return whether or not the control point is subscribed to the given service

Return Value: `UPNP_TRUE` if subscribed, UPNP_FALSE otherwise

Parameters:

`deviceHandle`	handle returned by UPnP_ControlOpenDevice
`serviceId`	id of service

See Also: UPnP_ControlSubscribe, UPnP_ControlUnsubscribe

1.37

void **UPnP_ControlSetServiceSubscriptionExpireWarning** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, UPNP_INT32 warningMsec)

Sets the time offset before subscription expiration at which a warning event will be generated.

This function should be called, if desired, before subscribing to the given service.

Return Value:	`nothing`	
Parameters:	`deviceHandle`	handle returned by UPnP_ControlOpenDevice
	`serviceId`	id of service
	`warningMsec`	time offset to generatewarning, in milliseconds
See Also:	UPnP_ControlSubscribe, UPnP_ControlUnsubscribe	

1.38

int **UPnP_ControlUnsubscribe** (UPnPControlDeviceHandle deviceHandle, UPNP_CHAR* serviceId, void* userData, UPNP_BOOL waitForCompletion)

Cancel a subscribed service

If waitForCompletion is UPNP_TRUE, this function waits for the unsubscribe operation to complete before returning. Otherwise, it returns immediately and a UPNP_CONTROL_EVENT_SUBSCRIPTION_CANCELLED event is generated when the operation completes.

Generates: UPNP_CONTROL_EVENT_SUBSCRIPTION_CANCELLED

Return Value:	`error` code	
Parameters:	`deviceHandle`	handle returned by UPnP_ControlOpenDevice
	`serviceId`	id of service to unsubscribefrom
	`userData`	passed to callback asuserRequestData for generatedevents
	`waitForCompletion`	see above description

See Also:

2 Internal Library Documentation

Names

These functions are not at the API level and should not be called from code outside the UPnP library.

2.1 Discovery/SSDP

Names

2.1.1 SSDP_INT32
SSDP_ServerInit (SSDPServerContext* ctx,
SSDP_UINT8* ipAddr,
SSDP_INT16 ipType,
const SSDP_CHAR* serverId,
SSDPCallback cb,
void* cookie)
SSDP server initialization routine. ..... **??**

2.1.2 SSDP_INT32
SSDP_ServerAddToSelectList (SSDPServerContext* ctx,
RTP_FD_SET* readList,
RTP_FD_SET* writeList,
RTP_FD_SET* errList)
..... **??**

2.1.3 SSDP_BOOL

Reads all messages posted to the multicast address ..... **??**

2.1.10 int **_SSDP_ReadMSearchHeader** (void* request, HTTPSession* ptr, HTTPHeaderType type, const HTTP_CHAR* name, const HTTP_CHAR* value)

Extracts MX and St headers from a SSDP request ..... **??**

2.1.11 int **_SSDP_ReadNotifyHeader** (void* request, HTTPSession* ptr, HTTPHeaderType type, const HTTP_CHAR* name, const HTTP_CHAR* value)

Extracts MX and St headers from a SSDP request ..... **??**

2.1.12 SSDP_INT32 **SSDP_QueueSearchResponse** (SSDPServerContext* ctx, SSDPSearch* search, const SSDP_CHAR* targetLocation, const SSDP_CHAR* targetURN, SSDP_UINT32 targetTimeoutSec)

Queues a response to the response list based on its scheduled delivery time. ..... **??**

2.1.13 SSDP_INT32

SSDP_SearchProcessState (SSDPClientSearch* search, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList) ..... **??**

These functions are not at the API level and should not be called from code outside the UPnP library.

2.1.1

SSDP_INT32 **SSDP_ServerInit** (SSDPServerContext* ctx, SSDP_UINT8* ipAddr, SSDP_INT16 ipType, const SSDP_CHAR* serverId, SSDPCallback cb, void* cookie)

SSDP server initialization routine.

This routine starts up SSDP services by creating a UDP socket, getting the ssdp multicast membership for the socket and initlializing ssdp context.

Return Value: `error` code

Parameters:

`ctx`	pointer to an uninitialized SSDP context structure
`ipAddr`	IP address of the host to bind the UDP socket to, if a NULL issupplied, the UDP socket is bound to the local IP address
`ipType`	ip version 4 or ipversion 6
`serverId`	String holding platform name
`cb`	SSDP Callback routine
`cookie`	cookie(runtime) to be stored in ssdp context to be passedlater to ssdp callback

2.1.2

SSDP_INT32 **SSDP_ServerAddToSelectList** (SSDPServerContext* ctx, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

2.1.3

SSDP_BOOL **SSDP_ServerProcessState** (SSDPServerContext* ctx, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

2.1.4

void **SSDP_ServerDestroy** (SSDPServerContext* ctx
pointer to SSDP context* /)

2.1.5

SSDP_INT32 **SSDP_SendNotify** (SSDPServerContext* ctx, const SSDP_CHAR* notifyType, const SSDP_CHAR* notifySubType, const SSDP_CHAR* usn, const SSDP_CHAR* location, SSDP_UINT32* timeout)

Send a SSDP notification for the device or service.

This routine sends ssdp alive and bye-bye notifications on the mulicast address for devices and servces

Return Value:	`error`	code
Parameters:	`ctx`	pointer to SSDP context
	`notifyType`	the notification type (NT) string
	`notifySubType`	pointer to string containing NT subtype
	`usn`	pointer to string containing USN header
	`location`	pointer to string containing Location header
	`timeout`	pointer to max-age header value

2.1.6

SSDP_INT32 **SSDP_SendResponse** (SSDPServerContext* ctx, SSDPPendingResponse* response)

Deliver a responce to SSDP discovery request.

Deliver a responce to SSDP discovery request.

Return Value:	`error`	code
Parameters:	`ctx`	pointer to SSDP context
	`response`	buffer holding response information

2.1.7

SSDP_INT32 **_SSDP_ProcessOneRequest** (SSDPServerContext* ctx)

Process an incoming SSDP discovery request.

Processes a SSDP discovery request available during a period given by timeoutMsec milli seconds.

Return Value: `error` code
Parameters: `ctx` pointer to SSDP context

2.1.8

int **SSDP_ParseRequest** (SSDPServerContext* ctx, SSDPServerRequest* ssdpRequest)

Extract SSDP request.

Retrieves messages from the multicast address, if ssdp request is detected a buffer holding request information is populated

Return Value: `error` code
Parameters:
`ctx` pointer to SSDP context
`ssdpRequest` address of the SSDPServerEvent structure to fill up

2.1.9

int **SSDP_McastRead** (void* cookie, SSDP_UINT8* buffer, SSDP_INT32 min, SSDP_INT32 max)

Reads all messages posted to the multicast address

Reads all messages posted to the multicast address

Return Value:	`error`	code
Parameters:	`cookie`	internal cookie
	`buffer`	pointer to buffer contaning request message
	`min`	
	`max`	max size to be read

2.1.10

int **_SSDP_ReadMSearchHeader** (void* request, HTTPSession* ptr, HTTPHeaderType type, const HTTP_CHAR* name, const HTTP_CHAR* value)

Extracts MX and St headers from a SSDP request

Extracts MX and St headers from a SSDP request

Return Value:	`error`	code
Parameters:	`request`	request buffer to be populated
	`ptr`	current HTTP session
	`type`	HTTP header type
	`name`	holds the name of the header
	`value`	holds the value of the header

2.1.11

int **_SSDP_ReadNotifyHeader** (void* request, HTTPSession* ptr, HTTPHeaderType type, const HTTP_CHAR* name, const HTTP_CHAR* value)

Extracts MX and St headers from a SSDP request

Extracts MX and St headers from a SSDP request

Return Value:	`error`	code
Parameters:	`request`	request buffer to be populated
	`ptr`	current HTTP session
	`type`	HTTP header type
	`name`	holds the name of the header
	`value`	holds the value of the header

2.1.12

SSDP_INT32 **SSDP_QueueSearchResponse** (SSDPServerContext* ctx, SSDPSearch* search, const SSDP_CHAR* targetLocation, const SSDP_CHAR* targetURN, SSDP_UINT32 targetTimeoutSec)

Queues a response to the response list based on its scheduled delivery time.

Queues a response to the response list based on its scheduled delivery time. A random delivery time within targetTimeoutSec duration is calculated. This response is positioned in the list according to its scheduled delivery time.

Return Value:	`error`	code
Parameters:	`ctx`	pointer to SSDP context
	`search`	pointer to the buffer containingthe request information
	`targetLocation`	pointer to string containing Location header
	`targetURN`	pointer to string containing USN header
	`targetTimeoutSec`	max-age header value

2.1.13

SSDP_INT32 **SSDP_CheckPendingResponses** (SSDPServerContext* ctx, SSDP_UINT32 currentTimeMsec)

Delivers responses scheduled for delivery.

Scan the pending response list and deliver responses for which the scheduled time count is less than supplied time count.

Return Value:	`error`	code
Parameters:	`ctx`	pointer to SSDP context
	`currentTimeMsec`	time against which the scheduledtime is checked

2.1.14

void **SSDP_ProcessError** (SSDP_CHAR* errMsg)

Process SSDP Errors

Process SSDP Errors

Return Value:	`None`	
Parameters:	`errMsg`	error message string

2.1.15

SSDP_UINT32 **SSDP_RandMax** (SSDP_UINT32 mxLimit)

generates a random number between 0 and mxLimit

generates a random number between 0 and mxLimit

Return Value:	`error`	code
Parameters:	`mxLimit`	upper limit of the random number

2.1.16

int **SSDP_SearchInit** (SSDPClientSearch* search, HTTPManagedClient* httpClient, SSDP_INT16 ipType, SSDP_CHAR* searchType, SSDP_INT32 maxResponseTimeoutSec, SSDPSearchCallback callbackFn, void* callbackData)

2.1.17

void **SSDP_SearchDestroy** (SSDPClientSearch* search)

2.1.18

int **SSDP_SearchExecute** (SSDPClientSearch* search)

2.1.19

SSDP_INT32 **SSDP_SearchAddToSelectList** (SSDPClientSearch* search, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

2.1.20

SSDP_BOOL **SSDP_SearchProcessState** (SSDPClientSearch* search, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

2.2 Description

These functions are not at the API level and should not be called from code outside the UPnP library.

2.3 Control/SOAP

Names

2.3.1 SOAP_INT32

SOAP_ActionInit (SOAPAction* action,
HTTPManagedClient*
httpClient,
SOAP_INT16 ipType,
const SOAP_CHAR* destUri,
const SOAP_CHAR*
soapAction,
SOAP_CHAR* headerStr,
SOAP_INT32 headerLen,
SOAP_CHAR* bodyStr,
SOAP_INT32 bodyLen,
SOAPActionCallback
callbackFn,
void* callbackData) ..... **??**

2.3.2 SOAP_INT32

These functions are not at the API level and should not be called from code outside the UPnP library.

2.3.1

SOAP_INT32 **SOAP_ActionInit** (SOAPAction* action, HTTPManagedClient* httpClient, SOAP_INT16 ipType, const SOAP_CHAR* destUri, const SOAP_CHAR* soapAction, SOAP_CHAR* headerStr, SOAP_INT32 headerLen, SOAP_CHAR* bodyStr, SOAP_INT32 bodyLen, SOAPActionCallback callbackFn, void* callbackData)

desc::: this will be called each time a soap action is to be performed. This will open a http client, and send an POST, the request structure is filled and send to be upnp client manager to add the request to its list.

Return Value: `error` code

2.3.2

SOAP_INT32 **SOAP_ActionInitEx** (SOAPAction* action, HTTPManagedClient* httpClient, SOAP_INT16 ipType, URLDescriptor* postURL, URLDescriptor* baseURL, const SOAP_CHAR* soapAction, SOAP_CHAR* headerStr, SOAP_INT32 headerLen, SOAP_CHAR* bodyStr, SOAP_INT32 bodyLen, SOAPActionCallback callbackFn, void* callbackData)

desc::: this will be called each time a soap action is to be performed. This will open a http client, and send an POST, the request structure is filled and send to be upnp client manager to add the request to its list.

Return Value: `error` code

2.3.3

SOAP_INT32 **SOAP_CreateHttpSession** (SOAPAction* action)

/*

Return Value: `error` code

2.3.4

void **SOAP_ActionDestroy** (SOAPAction* action)

close the session ::

Return Value: `error` code

2.3.5

SOAP_INT32 **SOAP_ActionExecute** (SOAPAction* action)

2.3.6

SOAP_INT32 **SOAP_ActionAddToSelectList** (SOAPAction* action, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

desc:: add to select list, and return timeout milliseconds to select for

Return Value: `error` code

2.3.7

SOAP_BOOL **SOAP_ActionProcessState** (SOAPAction* action, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

2.3.8

SOAP_INT32 **SOAP_SendActionRequest** (SOAPAction* action)

/*

Return Value: `error` code

2.3.9

SOAP_INT32 **SOAP_ReadActionResponse** (SOAPAction* action)

/*

Return Value: `error` code

2.3.10

SOAPActionState **_SOAP_ParseActionRespose** (HTTPResponseInfo* info, HTTPManagedClientSession* session, SOAPActionResponse* response)

2.4

Eventing/GENA

Names

GENA_SubscribeRequestInit (GENAClientRequest* request, HTTPManagedClient* httpClient, GENA_INT16 ipType, GENA_CHAR* relUrlStr, URLDescriptor* baseURL, GENA_CHAR* callbackUrl, GENA_INT32 timeoutSec, GENAClientRequestCallback callbackFn, void* callbackData) ..... **??**

2.4.6 GENA_INT32

GENA_RenewRequestInit (GENAClientRequest* request, HTTPManagedClient* httpClient, GENA_INT16 ipType, GENA_CHAR* serverUrl, URLDescriptor* baseURL, GENA_CHAR* sid, GENA_INT32 timeoutSec, GENAClientRequestCallback callbackFn, void* callbackData) ..... **??**

2.4.7 GENA_INT32

These functions are not at the API level and should not be called from code outside the UPnP library.

2.4.1

GENA_INT32 **GENA_ClientInit** (GENAClientContext* ctx, GENAClientCallback callback, void* cookie)

2.4.2

void **GENA_ClientDestroy** (GENAClientContext* ctx)

2.4.3

GENA_INT32 **GENA_ClientProcessEvent** (GENAClientContext* ctx, HTTPServerRequestContext* srv, HTTPSession* session, HTTPRequest* request, RTP_NET_ADDR* notifierAddr)

2.4.4

GENA_INT32 **GENA_SendEventResponse** (GENAClientEvent* genaEvent, HTTPServerRequestContext* srv, HTTPSession* session, HTTPRequest* request)

2.4.5

GENA_INT32 **GENA_SubscribeRequestInit** (GENAClientRequest* request, HTTPManagedClient* httpClient, GENA_INT16 ipType, GENA_CHAR* relUrlStr, URLDescriptor* baseURL, GENA_CHAR* callbackUrl, GENA_INT32 timeoutSec, GENAClientRequestCallback callbackFn, void* callbackData)

desc::: this will be called each time a GENA request is to be performed. This will open a http client, and send an POST, the request structure is filled and send to be upnp client manager to add the request to its list.

Return Value: `error` code

2.4.6

GENA_INT32 **GENA_RenewRequestInit** (GENAClientRequest* request, HTTPManagedClient* httpClient, GENA_INT16 ipType, GENA_CHAR* serverUrl, URLDescriptor* baseURL, GENA_CHAR* sid, GENA_INT32 timeoutSec, GENAClientRequestCallback callbackFn, void* callbackData)

desc::: this will be called each time a GENA request is to be performed. This will open a http client, and send an POST, the request structure is filled and send to be upnp client manager to add the request to its list.

Return Value: `error` code

2.4.7

GENA_INT32 **GENA_UnsubscribeRequestInit** (GENAClientRequest* request, HTTPManagedClient* httpClient, GENA_INT16 ipType, GENA_CHAR* serverUrl, URLDescriptor* baseURL, GENA_CHAR* sid, GENAClientRequestCallback callbackFn, void* callbackData)

desc::: this will be called each time a GENA request is to be performed. This

will open a http client, and send an POST, the request structure is filled and send to be upnp client manager to add the request to its list.

Return Value: `error` code

2.4.8

GENA_INT32 **GENA_CreateHttpSession** (GENAClientRequest* request, HTTPManagedClient* httpClient, GENA_CHAR* relUrlStr, URLDescriptor* baseURL, GENA_INT16 ipType)

/*

Return Value: `error` code

2.4.9

void **GENA_ClientRequestDestroy** (GENAClientRequest* request)

2.4.10

GENA_INT32 **GENA_ClientRequestExecute** (GENAClientRequest* request)

2.4.11

GENA_INT32 **GENA_ClientRequestAddToSelectList** (GENAClientRequest* request, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

desc:: add to select list, and return timeout milliseconds to select for

Return Value: `error` code

2.4.12

GENA_BOOL **GENA_ClientRequestProcessState** (GENAClientRequest* request, RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errList)

2.4.13

GENA_INT32 **GENA_SendRequest** (GENAClientRequest* request)

/*

Return Value: `error` code

2.4.14

GENA_INT32 **GENA_ReadResponse** (GENAClientRequest* request)

/*

Return Value: `error` codeIMPORTANT - response.sid is allocated memory in read header callback, it needs to be freed

2.4.15

GENA_INT32 **_GENA_ReadHeader** (void* userData, HTTPSession* session, HTTPHeaderType type, const GENA_CHAR* name, const GENA_CHAR* value)

/*

Return Value: `error` code