



RFC 32 - UPnP Device Category Specification

Confidential, Draft
RFC 0032

29 Pages

Abstract

Universal Plug and Play (UPnP) is a device interoperability standard that leverages IP and XML technologies to allow simple interaction between intelligent networked devices.

The goal of this RFC is to define an OSGi service to interoperate with UPnP compliant devices.

It allows the import of networked UPnP Devices to be accessible via other OSGi services through the mechanisms of the OSGi Device Access Architecture, as well as exporting of generic OSGi services as virtual UPnP devices to the outside world.

Bridging legacy devices into UPnP networks and vice-versa is one possible application of this API.

Copyright © Gatespace AB 2002.

This contribution is made to the Open Services Gateway Initiative (OSGI) as MEMBER LICENSED MATERIALS pursuant to the terms of the OSGI membership agreement and specifically the license rights and warranty disclaimers as set forth in Sections 3.2 and 12.1, respectively.

All company, brand and product names contained within this document may be trademarks that are the sole property of the respective owners.

The above notice must be included on all copies of this document that are made.



0 Document Information

0.1 Table of Contents

0 Document Information	2
0.1 Table of Contents	2
0.2 Status	3
0.3 Requirements	3
0.4 Terminology and Document Conventions	3
0.4.1 Service	3
0.4.2 Device	4
0.5 Revision History	4
1 Introduction	5
2 Motivation and Rationale	5
3 Technical Discussion	6
3.1 API Style	6
3.2 Exported Vs. Imported Devices	6
3.3 Granularity of Devices	6
3.4 Event API	7
3.5 State Variables	7
3.6 Icons	9
3.7 Presentation URL	9
3.8 Localization	9
3.9 Configuration	10
4 Networking considerations	10
4.1 UPnP multicasts	10
5 API Specification	10
5.1 org.osgi.service.upnp.UPnPDevice	10
5.1.1 Field Detail	11
5.1.2 Method Detail	15
5.2 org.osgi.service.upnp.UPnPService	16
5.2.1 Field Details	17
5.2.2 Method Details	17
5.3 org.osgi.service.upnp.UPnPAction	19
5.3.1 Method Detail	19
5.4 org.osgi.service.upnp.UPnPStateVariable	21
5.4.1 Field Detail	21
5.4.2 Method Detail	24
5.5 org.osgi.service.upnp.UPnPIcon	25
5.5.1 Method Detail	25



5.6 org.osgi.service.upnpn.UPnPEventListener	26
5.6.1 Field Detail	27
5.6.2 Method Detail	27
6 Security Considerations.....	28
6.1 Basic Limitations	28
7 Document Support	28
7.1 References	28
7.2 Author's Address	28
7.3 Acronyms and Abbreviations	29
7.4 End of Document.....	29

0.2 Status

This document specifies APIs for integrating UPnP devices with the Open Services Gateway Initiative. Discussion and suggestions for improvements are requested; distribution of this document is unlimited within OSGi.

0.3 Requirements

This document is a proposal for review by the Device Expert Group, and as such assumes working knowledge of the UPnP 1.0 and the OSGi 2.0 Device Access Specification. Many of the features described in the Device Access Specification provide a seamless integration with application bundles.

Because the UPnP protocol is based on IP communication, every device participating in an UPnP network must have basic IP connectivity. This requires support and configuration on the operating system layer.

UPnP recognizes mechanisms and policies on how to obtain an IP-address for a newly connected device. If there is no DHCP server available, an IP address is obtained through AUTO-IP. These basic mechanisms are to be handled by the operating system, and are outside the scope of the following API definitions.

Also, in the case where the OSGi UPnP implementation acts as a control point the tasks of assigning IP addresses to other devices on the network is to be handled by the operating system independent of the UpnP implementation.

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

One challenge of this RFC is to map the concepts of OSGi devices and services, as described in RFC 8 and the framework specification, to the concepts of UPnP devices and services.

Where the distinction between OSGi and UPnP services and devices is not obvious, such services and devices will be explicitly identified as either OSGi or UPnP.

0.4.1 Service

A OSGi service is a Java Object registered in the framework registry and described by a Java interface.

A UPnP service is a fundamental, controllable entity of a device, comparable to a command, that is described by XML.

0.4.2 Device

According to RFC 8, an OSGi device is a special type of service.

According to the UPnP definition, a device is a logical container of UPnP services and other UPnP devices. There are root devices that can contain a tree of other nested devices and services.

0.5 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial draft	9/12/01	Christian Kurzke, Gatespace Inc., chris@gatespace.com Tommy Bohlin, Gatespace AB, tommy@gatespace.com
Updated version	5/18/02	Christian Kurzke, Gatespace Inc., chris@gatespace.com
Minor update	7/5/02	Per Gustafson, Gatespace AB, perg@gatespace.com
Date Issue Update	8/9/02	Per Gustafson, Gatespace AB, perg@gatespace.com UpnPDate removed
Review Update 1	8/13/02	Per Gustafson, Gatespace AB, perg@gatespace.com Corrections of some errors in the javadoc. Added missing javadoc for UPnPEventListener.
Voting candidate 1	8/16/02	Christian Kurzke, Gatespace Inc., chris@gatespace.com Per Gustafson, Gatespace AB, perg@gatespace.com -Clarifications and corrections. -Added a property to be standardized for ConfigurationAdmin. -Fixed inconsistencies between document and java sources.
Voting candidate 2	9/11/02	Per Gustafson, Gatespace AB, perg@gatespace.com -Changed type of ConfigurationAdmin property from String to String[] after feedback from Peter Kriens. -Fixed copyright -Fixed minor typos

1 Introduction

The goal of this RFC is to specify an API that can be used to develop a set of OSGi bundles that interoperate with UPnP devices and UPnP control points.

Universal Plug and Play (UPnP) is a device interoperability standard that leverages IP and XML technologies to allow simple interaction between intelligent networked devices. An OSGi Framework provides a platform for running services, while UPnP provides a connectivity path between these services and other appliances. An OSGi Framework can simultaneously support many networks, including UPnP.

A UPnP base driver bundle can act as a UPnP control point, discover UPnP devices on the network, and map each device into an OSGi service that can be controlled by other bundles. The UPnP bundle can also detect Framework services of a certain type, export them, and present the services as virtual UPnP devices on the network for other hosts to discover and control.

UPnP devices and control point applications in the form of bundles can be developed with minimal effort.

Note: UPnP's ability to assign IP addresses to new devices on the network or do an auto-IP self-configuration should be handled at the operating system level; such functions are outside the scope of this RFC.

2 Motivation and Rationale

Universal Plug and Play (UPnP) is an open network architecture that is designed to enable simple, ad hoc communication among distributed devices and services from many vendors. This document contains information required to model and implement UPnP devices and control points.

UPnP leverages Internet protocols, including IP, TCP, UDP, HTTP, and XML. Like the Internet, contracts are based on wire protocols that are declarative, expressed in XML, and communicated via HTTP.

This RFC specifies an API that allows an OSGi bundle to access UPnP devices available on the network transparently, i.e., like any other local OSGi device service. A base driver implementation corresponding to this RFC as well as to RFC 8 is required.

This RFC also details how OSGi services can also be exported as "virtual" UPnP devices to the local network in a way that is transparent to normal UPnP devices. This can be used to bridge legacy devices to UPnP networks.

3 Technical Discussion

The central design decisions are as follows:

3.1 API Style

The proposed API follows correct object orientation style, with methods on the respective objects they operate on.

A downside of this style is that bundles that create virtual UPnP device service to be exported by the UPnP Service must implement some number of helper classes. Although a straightforward task, it can be tedious and error prone for the bundle programmer.

This RFC concentrates on providing a complete and correct API supporting all features of UPnP. Refinements intended to simplify the exporting issue will be left to a later revision of this document or to a separate RFC.

For now it is assumed that helper and convenience classes conforming to this RFC are supplied by a 3rd party.

3.2 Exported Vs. Imported Devices

The implementation goal is that exported and imported UPnP devices result in the same internal representation for identical devices. On one OSGi gateway, no difference should exist between an exported virtual UPnP device and an the imported virtual service representation of a physical UPnP device.

Application bundles interact with imported and exported representations of the device in the same way, simplifying the implementation of UPnP exporting and controlling bundles.

Imported and exported UPnP devices implement the same Java interface, and are nearly identically represented except for two marker properties that may be added to the service registration. One marker, `DEVICE_CATEGORY` should normally be set only on imported devices. The other marker, `UPNP_EXPORT`, should be set only on internally created devices, which are intended to be exported.

By not setting `DEVICE_CATEGORY` on internal UPnP devices, the Device Manager will not try to refine these devices. If the device service does not implement `Device` and does not have the `DEVICE_CATEGORY` property, it is not considered a "device" service according to the Device Access Specification.

By not setting `UPNP_EXPORT` on imported devices, the UPnP driver can avoid exporting its own imported devices and thus avoid creating an infinite loop. It can also be useful to not set `UPNP_EXPORT` on some internally created devices. This allows UPnP devices to be simulated within an OSGi gateway without announcing all of these devices to any networks.

3.3 Granularity of Devices

One UPnP root device could be mapped to one OSGi service, but this is too coarse. More likely, sub-devices are interesting for application bundles. If several sub devices are modeled as one OSGi service, it makes it harder to selectively use only one sub device.

Device properties are defined per UPnP sub device. Properties are retrievable via API calls on the UPnP device service; some are attached explicitly as OSGi service registry properties so they can be used in filter expressions when searching for registered UPnP device services. .

For external UPnP devices being imported and represented as OSGi services on a sub device level, it must be possible for an application to determine the original UPnP service hierarchy. The same is true, when the UPnP driver service attempts to export a number of virtual UPnP devices, it must be able to construct the tree of parent- and sub-devices.

The way to determine the OSGi device/ UPnP service hierarchy is by examining the service properties "UDN" (see 5.1.1.8), PARENT_UDN and CHILDREN_UDN (see 5.1.1.21 and 5.1.1.22)

3.4 Event API

UPnP events are sent using the whiteboard model, where a bundle interested in receiving UPnP events registers an object implementing the `UPnPEventListenerService` interface. To limit the events to be notified for, a filter property can be set.

If the service is registered with a property named "upnp.filter" with the value of an instance of an `org.osgi.framework.Filter` object, the listener will only be notified for matching events.

The available keywords used in the filter expression to describe the events are:

- **Device identity (UPnPDevice.UDN)**
Only events generated by services contained in the specific device are delivered.
- **Device type (UPnPDevice.TYPE)**
Only events generated by services hosted by device conforming to this given type are delivered.
- **Service identity (UPnPService.ID)**
Only events generated by services matching the given service ID are delivered.
- **Service type (UPnPService.TYPE)**
Only events generated by services of the specified type are delivered

If an event is generated, the services `notifyUPnPEvent(Dictionary events)` method is called. As a parameter, one or multiple events are passed. The Dictionary holds a pair of an `UPnPStateVariable`, which triggered the event, and an Object for the new value of the variable.

Special care has to be taken with the initial subscription to events. According to the UPnP specification, when a client newly subscribes to be notified for events, the device sends out an event for each variable, indicating its current status. This behavior simplifies the synchronization of a device and an event driven client.

The UPnP driver uses the white board event notification model. Any bundle which wants to be notified of any UPnP device events registers a service implementing the `UPnPEventListener` interface (see section: 5.6)The UPnP driver will intercept the registration and call the notify method for all variables selected by the services filter. After this initial "event shower", which serves the purpose to synchronize all listeners with the current state of the device's variables, the listener services will be called for each incoming event for events from variables they track.

Note: the call to the listener service's notification method is to be done asynchronously.

3.5 State Variables

The `UPnPStateVariable` interface encapsulates the properties of an UPnP state variable. In addition to the properties defined by the UPnP specification, the state variable is also mapped to a Java data type. The Java data type is used when an event is generated for this state variable, and when an action is performed containing arguments related to this state variable. There must be a strict correspondence between the UPnP data type and the Java data type, so that bundles using a particular UPnP device profile can predict the precise Java data type.

The following table defines the correspondence between UPnP data types and Java data types.

UPnP data type	Java data type	Description
ui1	Integer	Unsigned 1 Byte int.
ui2	Integer	Unsigned 2 Byte int.
ui4	Long	Unsigned 4 Byte int.
i1	Integer	1 Byte int.
i2	Integer	2 Byte int.
i4	Integer	4 Byte int. Must be between -2147483648 and 2147483647
int	Integer	Integer number
r4	Float	4 Byte float. Same format as float. Must be between 3.40282347E+38 to 1.17549435E-38.
r8	Double	8 Byte float. Same format as float. Must be between -1.79769313486232E308 and -4.94065645841247E-324 for negative values, and between 4.94065645841247E-324 and 1.79769313486232E308 for positive values, i.e., IEEE 64-bit (8-Byte) double.
number	Double	Same as r8
fixed.14.4	Double	Same as r8 but no more than 14 digits to the left of the decimal point and no more than 4 to the right.
float	Float	Floating-point number. Mantissa (left of the decimal) and/or exponent may have a leading sign. Mantissa and/or exponent may have leading zeros. Decimal character in mantissa is a period, i.e., whole digits in mantissa separated from fractional digits by period. Mantissa separated from exponent by E. (No currency symbol.) (No grouping of digits in the mantissa, e.g., no commas.)
char	Character	Unicode string. One character long.
string	String	Unicode string. No limit on length.
date	Date (always 00:00 hours)	Date in a subset of ISO 8601 format without time data.
dateTime	Date (default time zone)	Date in ISO 8601 format with optional time but no time zone.
dateTime.tz	Date (adjusted to default time zone)	Date in ISO 8601 format with optional time and optional time zone.
time	Long (ms since midnight)	Time in a subset of ISO 8601 format with no date and no time zone.
time.tz	Long (ms since midnight, adjusted to default time zone, wrapping at 0 and 24*60*60*1000)	Time in a subset of ISO 8601 format with optional time zone but no date.
boolean	Boolean	True or false
bin.base64	byte[]	MIME-style Base64 encoded binary BLOB. Takes 3 Bytes, splits

		them into 4 parts, and maps each 6 bit piece to an octet. (3 octets are encoded as 4.) No limit on size. The Java byte array will hold the decoded content of the BLOB.
bin.hex	byte[]	Hexadecimal digits representing octets. Treats each nibble as a hex digit and encodes as a separate Byte. (1 octet is encoded as 2.) No limit on size. The Java byte array will hold the decoded content of the BLOB.
uri	String	Universal Resource Identifier.
uuid	String	Universally Unique ID. Hexadecimal digits representing octets. Optional embedded hyphens are ignored.

The function `QueryStateVariable` defined in the UPnP specification has been deprecated and is not implemented. Instead it is recommended to use the UPnP event mechanism (see section: 3.4) to track State Variables.

3.6 Icons

Each UPnP device can include an URI reference to an icon. This icon can either be served by an HTTP server running on the device itself or the URI can point to a different resource on the network.

When a bundle wants to access the icon of an imported UPnP device, the UPnP driver gets the data and presents it to the application through an input stream.

For a bundle, which wants to export a UPnP device and include an icon with the device, it must provide a class implementing the `UPnPIcon` interface and provide an input stream to the actual data. The UPnP driver bundle will then take care of registering the icon with an HTTP server and announcing the URI to the data with the rest of the UPnP device data.

Depending on the support for different locales, resolutions or color depths, a device might have more than one possible icon representing it.

In order to obtain localized icons, the method `getIcons(String locale)` can be used to obtain different versions. If the locale specified is a null argument, then the call will return the icons of the default locale of the called device (not the default locale of the control point).

3.7 Presentation URL

The presentation page is provided by the service property `PRESENTATION_URL`. The working assumption is that the exporting service registers its own servlet with the HTTP service to serve out this interface.

Another approach involves dynamically generating this interface from the UPnP service, based on the internal representation of the device, which is out of the scope for this release of the specification.

3.8 Localization

All values of the UPnP properties are obtained from the device using the device's default locale. If an application wants to query for a set of localized property values, it has to use the method `UPnPDevice.getDescriptions(String locale)`.

For localized versions of the icons, the method `UPnPDevice.getIcons(String locale)` is to be used.

3.9 Configuration

In order to provide a standardized way to configure a UPnP driver bundle, the following configuration management (CM) properties are defined:

upnp.ssdp.address

The value is a String[] with a list of IP addresses on the form aa.bb.cc.dd[:port]. Those addresses define the interfaces which the UPnP driver is operating on. If no port is specified, port 1900 is assumed as default.

If no ssdp address is specified, the default assumed will be 239.255.255.250:1900

4 Networking considerations

4.1 UPnP multicasts

It has to be made certain that the operating system supports multicasting on the selected Ethernet device. In certain cases, also a multicasting route has to be set in the operating system routing table.

Those setups are highly dependent on the underlying operating system and beyond the scope of this RFC.

5 API Specification

5.1 org.osgi.service.upnp.UPnPDevice

public interface **UPnPDevice**

Represents a UPnP device. For each UPnP root and embedded device, an object is registered with the framework under the UPnPDevice interface.

The relationship between a root device and its embedded devices can be deduced using the `UPnPDevice.CHILDREN_UDN` and `UPnPDevice.PARENT_UDN` service properties.

The values of the UPnP property names are defined by the Universal Plug and Play Forum.

All values of the UPnP properties are obtained from the device using the device's default locale. If an application wants to query for a set of localized property values, it has to use the method `UPnPDevice.getDescriptions(String locale)`.

5.1.1 Field Detail

5.1.1.1 *MATCH_GENERIC*

```
public static final int MATCH_GENERIC
```

Constant for the UPnP device match scale, indicating a generic match for the device.

Value: 1

5.1.1.2 *MATCH_TYPE*

```
public static final int MATCH_TYPE
```

Constant for the UPnP device match scale, indicating a match with the device type.

Value: 3

5.1.1.3 *MATCH_MANUFACTURER_MODEL*

```
public static final int MATCH_MANUFACTURER_MODEL
```

Constant for the UPnP device match scale, indicating a match with the device model.

Value: 7

5.1.1.4 *MATCH_MANUFACTURER_MODEL_REVISION*

```
public static final int MATCH_MANUFACTURER_MODEL_REVISION
```

Constant for the UPnP device match scale, indicating a match with the device revision.

Value: 15

5.1.1.5 *MATCH_MANUFACTURER_MODEL_REVISION_SERIAL*

```
public static final int MATCH_MANUFACTURER_MODEL_REVISION_SERIAL
```

Constant for the UPnP device match scale, indicating a match with the device revision and the serial number.

Value: 31

5.1.1.6 *DEVICE_CATEGORY*

```
public static final java.lang.String DEVICE_CATEGORY
```

Constant for the value of the service property `DEVICE_CATEGORY` used for all UPnP devices.

Value: "UPnP"

See: `org.osgi.service.device.Constants.DEVICE_CATEGORY`

5.1.1.7 *UPNP_EXPORT*

`public static final java.lang.String UPNP_EXPORT`

The `UPnP.export` service property is a hint that marks a device to be picked up and exported by the UPnP Service. Imported devices do not have this property set. The registered property requires no value.

Value: `"UPnP.export"`

5.1.1.8 *UDN*

`public static final java.lang.String UDN`

Name Constant for the Unique Device Name (UDN) property. It is the fundamental identifier of an instance of a `UPnPDevice`.

The value of the property is a String representation of the Device UDN.

Value: `"UPnP.device.UDN"`

Mandatory

5.1.1.9 *ID*

`public static final java.lang.String ID`

Name Constant for the Unique Device ID property. This property is an alias to `UPnPDevice.UDN`. It is merely provided for reasons of symmetry with the `UPnPService.ID` property.

The value of the property is a String representation of the Device UDN.

Value: `"UPnP.device.UDN"`

5.1.1.10 *TYPE*

`public static final java.lang.String TYPE`

Property key for the UPnP Device Type property.. Some standard property values are defined by the Universal Plug and Play Forum. The type string also includes a version number as defined in the UPnP specification. This service registration property must be set.

For standard devices defined by a UPnP Forum working committee, must consist of the following components in the given order separated by colons:

- `urn`
- `schemas-upnp-org`
- `device`
- a device type suffix
- an integer device version

For non-standard devices specified by UPnP vendors following components must be specified in the given order separated by colons:

- urn
- an ICANN domain name owned by the vendor
- device
- a device type suffix
- an integer device version

To allow for backward compatibility the UPnP import driver will automatically generate additional Device Type property entries for smaller versions than the current one. If for example a device announces its type as version 3, than properties for version 2 and 1 will be automatically generated.

In the case of exporting a UPnPDevice, the highest available version is being announced on the network.

Syntax Example: urn:schemas-upnp-org:device:deviceType:v

Value: "UPnP.device.type"

Mandatory

5.1.1.11 MANUFACTURER

public static final java.lang.String **MANUFACTURER**

Mandatory property key for the device manufacturer property. The property value holds a String representation of the device manufacturer's name.

Value: "UPnP.device.manufacturer"

Mandatory

5.1.1.12 MODEL_NAME

public static final java.lang.String **MODEL_NAME**

Mandatory property key for the device model name. The property value holds a String giving more information about the device model.

Value: "UPnP.device.modelName"

Mandatory

5.1.1.13 FRIENDLY_NAME

public static final java.lang.String **FRIENDLY_NAME**

Mandatory property key for a short user friendly version of the device name. The property value holds a String with the user friendly name of the device.

Value: "UPnP.device.friendlyName"

Mandatory

5.1.1.14 MANUFACTURER_URL

```
public static final java.lang.String MANUFACTURER_URL
```

Optional property key for a URL to the device manufacturers Web site. The value of the property is a String representing the URL.

Value: "UPnP.device.manufacturerURL"

Optional

5.1.1.15 MODEL_DESCRIPTION

```
public static final java.lang.String MODEL_DESCRIPTION
```

Optional (but recommended) property key for a String object with a long description of the device for the end user.

Value: "UPnP.device.modelDescription"

Recommended

5.1.1.16 MODEL_NUMBER

```
public static final java.lang.String MODEL_NUMBER
```

Optional (but recommended) property key for a String class typed property holding the model number of the device.

Value: "UPnP.device.modelNumber"

Recommended

5.1.1.17 MODEL_URL

```
public static final java.lang.String MODEL_URL
```

Optional property key for a String typed property holding a string representing the URL to the Web site for this model.

Value: "UPnP.device.modelURL"

Optional

5.1.1.18 SERIAL_NUMBER

```
public static final java.lang.String SERIAL_NUMBER
```

Optional (but recommended) property key for a String typed property holding the serial number of the device.

Value: "UPnP.device.serialNumber"

Recommended

5.1.1.19 *UPC*

```
public static final java.lang.String UPC
```

Optional property key for a String typed property holding the Universal Product Code (UPC) of the device.

Value: "UPnP.device.UPC"

Optional

5.1.1.20 *PRESENTATION_URL*

```
public static final java.lang.String PRESENTATION_URL
```

Optional (but recommended) property key for a String typed property holding a string representing the URL to a device representation Web page.

Value: "UPnP.presentationURL"

Recommended

5.1.1.21 *PARENT_UDN*

```
public static final java.lang.String PARENT_UDN
```

The property key that must be set for all embedded devices. It contains the UDN of the parent device. The property is not set for root devices.

The property is string valued. **Value:** "UPnP.device.parentUDN"

5.1.1.22 *CHILDREN_UDN*

```
public static final java.lang.String CHILDREN_UDN
```

The property key that must be set for all devices containing other embedded devices.

The value is an array of UDNs for each of the device's children (String[]). The array contains UDNs for the immediate descendants only.

If an embedded device in turn contains embedded devices, the latter are not included in the array.

The UPnP Specification does not encourage more than two levels of nesting.

The property is not set if the UPnP device does not contain any embedded devices.

The property is String[] valued.

Value: "UPnP.device.children.UDN"

5.1.2 Method Detail

5.1.2.1 *getService*

```
public UPnPService getService(java.lang.String serviceId)
```

Locates a specific service by serviceId.

Parameters:

serviceId - the service id

Returns:

The requested service or null if not found.

5.1.2.2 *getServices*

```
public UPnPService[] getServices()
```

Lists all services provided by this device.

Returns:

Array of services or null if no services are available.

5.1.2.3 *getIcons*

```
public UPnPIcon[] getIcons(java.lang.String locale)
```

Lists all icons for this device in a given locale. The UPnP spec allows a device to present different icons based on the clients locale.

Parameters:

locale - A language tag as defined by RFC 1766 and maintained by ISO 639. Examples include "de", "en" or "en-US". The default locale of the device is specified by passing a null argument.

Returns:

Array of icons or null if no icons are available.

5.1.2.4 *getDescriptions*

```
public java.util.Dictionary getDescriptions(java.lang.String locale)
```

Get a set of localized UPnP properties. The UPnP specification allows a device to present different device properties based on the client's locale.

The properties used to register the UPnPDevice Service in the OSGi registry are based on the device's default locale. To obtain a localized set of the properties, an application can use this method.

Note: Not all properties might be available in all locales. This method does **not** substitute missing properties with their default locale versions.

Parameters:

locale - A language tag as defined by RFC 1766 and maintained by ISO 639. Examples include "de", "en" or "en-US". The default locale of the device is specified by passing a null argument.

Returns:

Dictionary mapping property name Strings to property value Strings

5.2 org.osgi.service.upnp.UPnPService

```
public interface UPnPService
```

A UPnP service. Each UPnP device contains zero or more services. The UPnP description for a service defines actions, their arguments, and event characteristics.

5.2.1 Field Details

5.2.1.1 TYPE

`public static final java.lang.String TYPE`

Property key for the optional service type uri.

The Service type property is used when registering UPnPDevice services and UPnPEventListener services. The value of the property contains a String array (String[]) of service types. A UPnPDevice service can thus announce what types of services it contains. A UPnPEventListener can announce for what type of UPnP services it wants notification.

The service version is encoded in the type string as specified in the UPnP specification.

A null value as the property value is a wildcard, matching **all** service types.

Value: "UPnP.service.type"

Optional

See Also:

[getType\(\)](#)

5.2.1.2 ID

`public static final java.lang.String ID`

Property key for the optional service id.

The Service id property is used when registering UPnPDevice services and UPnPEventListener services. The value of the property contains a String array (String[]) of service ids. A UPnPDevice service can thus announce what service IDs it contains. A UPnPEventListener can announce for what UPnP service ids it wants notification.

A service id does **not** have to be universally unique. It must be unique only within a device.

A null value as the property value is a wildcard, matching all services.

Value: "UPnP.service.id"

Optional

5.2.2 Method Details

5.2.2.1 getId

`public java.lang.String getId()`

Returns the `serviceId` field in the UPnP service description.

For standard services defined by a UPnP Forum working committee, the `serviceId` must contain the following components in the indicated order:

- `urn:upnp-org:serviceId:`
- service ID suffix

Example: `urn:upnp-org:serviceId:serviceID`.

Note that `upnp-org` is used instead of `schemas-upnp-org` in this example because an XML schema is not defined for each `serviceId`.

For non-standard services specified by UPnP vendors, the service id must contain the following components in the indicated order:

- urn:
- ICANN domain name owned by the vendor
- :serviceId:
- service ID suffix

Example: urn:domain-name:serviceId:serviceID.

Returns:

The service ID suffix defined by a UPnP Forum working committee or specified by a UPnP vendor. Must be <= 64 characters. Single URI.

5.2.2.2 *getType*

```
public java.lang.String getType()
```

Returns the `serviceType` field in the UPnP service description.

For standard services defined by a UPnP Forum working committee, the `serviceType` must contain the following components in the indicated order:

- urn:schemas-upnp-org:service:
- service type suffix:
- integer service version

Example: urn:schemas-upnp-org:service:serviceType:v.

For non-standard services specified by UPnP vendors, the `serviceType` must contain the following components in the indicated order:

- urn:
- ICANN domain name owned by the vendor
- :service:
- service type suffix:
- integer service version

Example: urn:domain-name:service:serviceType:v.

Returns:

The service type suffix defined by a UPnP Forum working committee or specified by a UPnP vendor. Must be <= 64 characters, not including the version suffix and separating colon. Single URI.

5.2.2.3 *getVersion*

```
public java.lang.String getVersion()
```

Returns the version suffix encoded in the `serviceType` field in the UPnP service description.

Returns:

The integer service version defined by a UPnP Forum working committee or specified by a UPnP vendor.

5.2.2.4 *getAction*

public [UPnPAction](#) **getAction**(java.lang.String name)

Locates a specific action by name. Looks up an action by its name.

Parameters:

name - Name of action. Must not contain hyphen or hash characters. Should be < 32 characters.

Returns:

The requested action or `null` if no action is found.

5.2.2.5 *getActions*

public [UPnPAction](#)[] **getActions**()

Lists all actions provided by this service.

Returns:

Array of actions or `null` if no actions are defined for this service.

5.2.2.6 *getStateVariables*

public [UPnPStateVariable](#)[] **getStateVariables**()

Lists all state variables provided by this service.

Returns:

Array of state variables or `null` if none are defined for this service.

5.2.2.7 *getStateVariable*

public [UPnPStateVariable](#) **getStateVariable**(java.lang.String name)

Gets a state variables provided by this service by name

Parameters:

name - Name of the State Variable

Returns:

State variables or `null` if no such state variable exists for this service.

5.3 org.osgi.service.upnp.UPnPAction

public interface **UPnPAction**

A UPnP action. Each UPnP service contains zero or more actions. Each action may have zero or more UpnP state variables as arguments.

5.3.1 Method Detail

5.3.1.1 *getName*

public java.lang.String **getName**()

Returns the action name. The action name corresponds to the `name` field in the `actionList` of the service description.

- For standard actions defined by a UPnP Forum working committee, action names must not begin with `X_` nor `A_`.
- For non-standard actions specified by a UPnP vendor and added to a standard service, action names must begin with `X_`.

Returns:

Name of the action. Must not contain a hyphen character or a hash character

5.3.1.2 *getReturnArgumentName*

```
public java.lang.String getReturnArgumentName()
```

Returns the name of the designated return argument.

One of the output arguments can be flagged as a designated return argument.

Returns:

The name of the designated return argument or `null` if none is marked.

5.3.1.3 *getInputArgumentNames*

```
public java.lang.String[] getInputArgumentNames()
```

Lists all input arguments for this action.

Each action may have zero or more input arguments.

Returns:

Array of input argument names or `null` if no input arguments.

See Also:

[UPnPStateVariable](#)

5.3.1.4 *getOutputArgumentNames*

```
public java.lang.String[] getOutputArgumentNames()
```

List all output arguments for this action.

Returns:

Array of output argument names or `null` if there are no output arguments.

See Also:

[UPnPStateVariable](#)

5.3.1.5 *getStateVariable*

```
public UPnPStateVariable getStateVariable(java.lang.String argumentName)
```

Finds the state variable associated with an argument name. Helps to resolve the association of state variables with argument names in UPnP actions.

Parameters:

`argumentName` – The name of the action argument

Returns:

State variable associated with the named argument or `null` if there is no such argument.

See Also:

[UPnPStateVariable](#)

5.3.1.6 *invoke*

```
public java.util.Dictionary invoke(java.util.Dictionary args)
throws java.lang.Exception
```

Invokes the action. The input and output arguments are both passed as dictionaries. Each dictionary entry has a `String` key representing the argument name and the value is the argument itself. The class of an argument value must match the class specified by the `UPnPStateVariable` associated with the argument. The input argument dictionary must contain exactly those arguments listed by `getInputArguments`. The output argument dictionary will contain exactly those arguments listed by `getOutputArguments`.

Parameters:

`args` – A Dictionary of arguments. Must contain the correct set and type of arguments for this action. May be null if no input arguments exist.

Returns:

A Dictionary with the output arguments. null if the action has no output arguments.

Throws:

`java.lang.Exception` - If the execution fails for any reason.

See Also:

[UPnPStateVariable](#)

5.4 **org.osgi.service.upnp.UPnPStateVariable**

```
public interface UPnPStateVariable
```

The meta-information of a UPnP state variable as declared in the device's service state table (SST).

Method calls to interact with a device (e.g. `UPnPAction.invoke(...)`) use this class to encapsulate meta information about the input and output arguments.

The actual values of the arguments are passed as Java objects. The mapping of types from UPnP data types to Java data types is performed according to the table in section 3.5 .

5.4.1 **Field Detail**

5.4.1.1 *TYPE_UI1*

```
public static final java.lang.String TYPE_UI1
    Value: "ui1"
```

5.4.1.2 *TYPE_UI2*

```
public static final java.lang.String TYPE_UI2
    Value: "ui2"
```

5.4.1.3 *TYPE_UI4*

```
public static final java.lang.String TYPE_UI4
    Value: "ui4"
```

**5.4.1.4 TYPE_I1**

```
public static final java.lang.String TYPE_I1
    Value: "i1"
```

5.4.1.5 TYPE_I2

```
public static final java.lang.String TYPE_I2
    Value: "i2"
```

5.4.1.6 TYPE_I4

```
public static final java.lang.String TYPE_I4
    Value: "i4"
```

5.4.1.7 TYPE_INT

```
public static final java.lang.String TYPE_INT
    Value: "int"
```

5.4.1.8 TYPE_R4

```
public static final java.lang.String TYPE_R4
    Value: "r4"
```

5.4.1.9 TYPE_R8

```
public static final java.lang.String TYPE_R8
    Value: "r8"
```

5.4.1.10 TYPE_NUMBER

```
public static final java.lang.String TYPE_NUMBER
    Value: "number"
```

5.4.1.11 TYPE_FIXED_14_4

```
public static final java.lang.String TYPE_FIXED_14_4
    Value: "fixed.14.4"
```

5.4.1.12 TYPE_FLOAT

```
public static final java.lang.String TYPE_FLOAT
    Value: "float"
```

5.4.1.13 TYPE_CHAR

```
public static final java.lang.String TYPE_CHAR
    Value: "char"
```

5.4.1.14 TYPE_STRING

```
public static final java.lang.String TYPE_STRING
```



Value: "string"

5.4.1.15 TYPE_DATE

```
public static final java.lang.String TYPE_DATE
    Value: "date"
```

5.4.1.16 TYPE_DATETIME

```
public static final java.lang.String TYPE_DATETIME
    Value: "dateTime"
```

5.4.1.17 TYPE_DATETIME_TZ

```
public static final java.lang.String TYPE_DATETIME_TZ
    Value: "dateTime.tz"
```

5.4.1.18 TYPE_TIME

```
public static final java.lang.String TYPE_TIME
    Value: "time"
```

5.4.1.19 TYPE_TIME_TZ

```
public static final java.lang.String TYPE_TIME_TZ
    Value: "time.tz"
```

5.4.1.20 TYPE_BOOLEAN

```
public static final java.lang.String TYPE_BOOLEAN
    Value: "boolean"
```

5.4.1.21 TYPE_BIN_BASE64

```
public static final java.lang.String TYPE_BIN_BASE64
    Value: "bin.base64"
```

5.4.1.22 TYPE_BIN_HEX

```
public static final java.lang.String TYPE_BIN_HEX
    Value: "bin.hex"
```

5.4.1.23 TYPE_URI

```
public static final java.lang.String TYPE_URI
    Value: "uri"
```

5.4.1.24 TYPE_UUID

```
public static final java.lang.String TYPE_UUID
    Value: "uuid"
```

5.4.2 Method Detail

5.4.2.1 *getName*

```
public java.lang.String getName()
```

Returns the variable name.

- All standard variables defined by a UPnP Forum working committee must not begin with `x_` nor `A_`.
- All non-standard variables specified by a UPnP vendor and added to a standard service must begin with `x_`.

Returns:

Name of State Variable. Must not contain a hyphen character nor a hash character. Should be < 32 characters.

5.4.2.2 *getJavaDataType*

```
public java.lang.Class getJavaDataType()
```

Returns the Java class associated with the UPnP data type of this state variable.

Mapping between the UPnP data types and Java classes is performed according to the schema mentioned above.

Returns:

A class object corresponding to the Java type of this argument.

5.4.2.3 *getUPnPDataType*

```
public java.lang.String getUPnPDataType()
```

Returns the UPnP data type of this state variable. Valid types are defined as constants.

Returns:

The UPnP type of this state variable, as defined in above constants.

5.4.2.4 *getDefaultValue*

```
public java.lang.Object getDefaultValue()
```

Returns the default value, if defined.

Returns:

The default value or `null` if not defined. The type of the returned object can be determined by `getJavaDataType`.

5.4.2.5 *getAllowedValues*

```
public java.lang.String[] getAllowedValues()
```

Returns the allowed values, if defined. Allowed values can only be defined for String types.

Returns:

The allowed values or `null` if not defined. Should be less than 32 characters.

5.4.2.6 *getMinimum*

```
public java.lang.Number getMinimum()
```

Returns the minimum value, if defined. Minimum values can only be defined for numeric types.

Returns:

The minimum value or `null` if not defined.

5.4.2.7 *getMaximum*

```
public java.lang.Number getMaximum()
```

Returns the maximum value, if defined. Maximum values can only be defined for numeric types.

Returns:

The maximum value or `null` if not defined.

5.4.2.8 *getStep*

```
public java.lang.Number getStep()
```

Returns the size of an increment operation, if defined. Step sizes can only be defined for numeric types.

Returns:

The increment size or `null` if not defined.

5.4.2.9 *sendsEvents*

```
public boolean sendsEvents()
```

Tells if this `StateVariable` can be used as an event source. If the `StateVariable` is eventable, an event listener service can be registered to be notified when changes to the variable appear.

Returns: `true` if the `StateVariable` generates events, `false` otherwise.

5.5 **org.osgi.service.upnp.UPnPIcon**

```
public interface UPnPIcon
```

A UPnP icon representation. Each UPnP device can contain zero or more icons.

5.5.1 **Method Detail**

5.5.1.1 *getMimeType*

```
public java.lang.String getMimeType()
```

Returns the MIME type of the icon. This method returns the format in which the icon graphics readable from the `InputStream` obtained by `getInputStream()` is encoded.

The format of the returned string is in accordance to RFC2046. A list of valid MIME types is maintained by the IANA at <http://ftp.isi.edu/in-notes/iana/assignments/media-types/media-types>.

Typical values returned include: "image/jpeg" or "image/gif"

Returns:

The mime type of the encoded icon

5.5.1.2 *getWidth*

```
public int getWidth()
```

Returns the width of the icon in pixels.

If the actual width of the icon is unknown, a -1 is returned.

Returns:

The width in pixels, or -1 if unknown.

5.5.1.3 *getHeight*

```
public int getHeight()
```

Returns the height of the icon in pixels.

If the actual height of the icon is unknown, a -1 is returned.

Returns:

the height in pixels, or -1 if unknown.

5.5.1.4 *getSize*

```
public int getSize()
```

Returns the size of the icon in bytes.

This method returns the number of bytes of the icon available to read from the InputStream object obtained by `getInputStream()`.

If the actual size can not be determined, a -1 is returned.

Returns:

The icon size in bytes, or -1 if the size is unknown.

5.5.1.5 *getDepth*

```
public int getDepth()
```

Returns the color depth of the icon in bits.

Returns:

The color-depth in bits. If the actual color depth of the icon is unknown, -1 is returned.

5.5.1.6 *getInputStream*

```
public java.io.InputStream getInputStream()  
throws java.io.IOException
```

Returns an InputStream for the icon data.

The InputStream provides a way for a client to read the actual icon graphics data. The number of bytes available from this InputStream can be determined via `getSize()`

The format of the data encoded can be determined by the MIME type available via `getMimeType()`

Returns:

An InputStream to read the icon graphics data from.

Throws:

`java.io.IOException` -

See Also:

[`getMimeType\(\)`](#)

5.6 **org.osgi.service.upnpn.UPnPEventListener**

```
public interface UPnPEventListener
```

UPnP Events are mapped and delivered to applications according to the OSGi whiteboard model. An application that wishes to be notified of events generated by a particular UPnP Device registers a service extending this interface.

The notification call from the UPnP Service to any `UPnPEventListener` object must be done asynchronous with respect to the originator (in a separate thread).

Upon registration of the UPnP Event Listener service with the Framework, the service is notified for each variable which it listens for with an initial event containing the current value of the variable. Subsequent notifications only happen on changes of the value of the variable.

A UPnP Event Listener service filter the events it receives. This event set is limited using a standard framework filter expression which is specified when the listener service is registered.

The filter is specified in a property named "upnp.filter" and has as a value an object of type `org.osgi.framework.Filter`.

When the Filter is evaluated, the following keywords are recognized as defined as literal constants in the `UPnPDevice` class.

The valid subset of properties for the registration of UPnP Event Listener services are:

- `UPnPDevice.TYPE` -- Which type of device to listen for events.
- `UPnPDevice.ID` -- The ID of a specific device to listen for events.
- `UPnPService.TYPE` -- The type of a specific service to listen for events.
- `UPnPService.ID` -- The ID of a specific service to listen for events.

5.6.1 Field Detail

5.6.1.1 *UPNP_FILTER*

```
public static final java.lang.String UPNP_FILTER
```

Key for a service property having a value that is an object of type `org.osgi.framework.Filter` and that is used to limit received events.

Value: "upnp.filter"

See Also: [Constant Field Values](#)

5.6.2 Method Detail

5.6.2.1 *notifyUPnPEvent*

```
public void notifyUPnPEvent(java.lang.String deviceId,  
                           java.lang.String serviceId,  
                           java.util.Dictionary events)
```

Callback method that is invoked for received events. The events are collected in a `Dictionary` object. Each entry has a `String` key representing the event name (= state variable name) and the new value of the state variable. The class of the value object must match the class specified by the UPnP State Variable associated with the event. This method must be called asynchronously.

Parameters:

`deviceId` - ID of the device sending the events
`serviceId` - ID of the service sending the events
`events` - Dictionary object containing the new values for the state variables that has changed

6 Security Considerations

6.1 Basic Limitations

UPnP is based on HTTP and uses plain text XML messages to control devices. For this reason, it does not provide any inherent security mechanisms.

7 Document Support

7.1 References

1. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
2. Microsoft inc. 2000, Universal Plug and Play Device Architecture, http://www.upnp.org/download/UPnPDA10_20000613.htm
3. W3C, 2000, Simple Object Access Protocol 1.1, <http://www.w3.org/TR/SOAP/>

7.2 Author's Address

Name	Christian Kurzke
Company	Gatespace Inc.
Address	720 San Antonio Rd, Palo Alto, CA 94303
Voice	+1 650 575 0670
e-mail	chris@gatespace.com

Name	Tommy Bohlin
Company	Gatespace AB
Address	Stora Badhusgatan 18-20, SE-411 21 Göteborg, Sweden
Voice	+46 31 7439815
e-mail	tommy@gatespace.com

Name	Per Gustafson
Company	Gatespace AB
Address	Stora Badhusgatan 18-20, SE-411 21 Göteborg, Sweden
Voice	+46 31 7439823
e-mail	perg@gatespace.com

7.3 Acronyms and Abbreviations

7.4 End of Document